19980924 059

A DECISION THEORETIC APPROACH

FOR INTERFACE AGENT DEVELOPMENT

DISSERTATION
Scott Michael Brown
Captain, USAF

AFIT/DS/ENG/98-12

Approved for public release; distribution unlimited

DTIC QUALITY INSPECTED 1

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

AFIT/DS/ENG/98-12

A DECISION THEORETIC APPROACH
FOR INTERFACE AGENT DEVELOPMENT

DISSERTATION

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Scott Michael Brown, B.Comp.E., M.S.

Captain, USAF

September, 1998

A DECISION THEORETIC APPROACH
FOR INTERFACE AGENT DEVELOPMENT

Scott Michael Brown, B.Comp.E., M.S.

Captain, USAF

Approved:

Eugene Santos Jr.   18 Aug 98

Dr. Eugene Santos Jr. (Chairman)   Date

Martin R Stytz   31 aug 98

Dr. Martin R. Stytz   Date

Mark E. Oxley   28 Aug 98

Dr. Mark E. Oxley   Date

Alan V. Lair   31 Aug 98

Dr. Alan V. Lair (Dean's Representative)   Date

Robert A. Calico, Jr.
Dean, Graduate School of Engineering

Acknowledgements

Table of Contents

iv

List of Figures

AFIT/DS/ENG/98-12

Abstract

The complexity of current software applications is overwhelming users. The need exists for intelligent interface agents to address the problem of increasing taskload that is overwhelming the human user. Interface agents could help alleviate user taskload by extracting and analyzing relevant information, and providing information abstractions of that information, and providing timely, beneficial assistance to users. These agents could communicate with the user through the existing user interface and also adapt to user needs and behaviors.

Central to providing assistance to a user is the issue of correctly determining the user's intent. This dissertation presents an effective, efficient, and extensible decision-theoretic architecture for user intent ascription. The multi-agent architecture, the Core Interface Agent architecture, provides a dynamic, uncertainty-based knowledge representation for modeling the inherent ambiguity in ascribing user intent. The knowledge representation, a Bayesian network, provides an intuitive, mathematically sound way of determining the likelihood a user is pursuing a goal. This likelihood, combined with the utility of offering assistance to the user, provides a decision-theoretic approach to offering assistance to the user. The architecture maintains an accurate user model of the user's goals within a target system environment. The on-line maintenance of the user model is performed by a collection of correction adaptation agents. Because the decision-theoretic methodology is domain-independent, this new methodology for user intent ascription is readily extensible over new application domains. Furthermore, it also offers the ability to "bootstrap" intent understanding without the need for often lengthy and costly knowledge elicitation. Thus, as a side benefit, the process can mitigate the classic knowledge acquisition bottleneck problem.

The architecture's utility for providing assistance to users is demonstrated in three disparate systems — a probabilistic expert system shell, a virtual space plane, and a natural language interface information management system. Results indicate the CIA architecture is capable of providing useful assistance to users. This assistance is tailored to individual users based on the tasks they are performing, their preferences, biases, abilities, and application usage patterns. Furthermore, the assistance can be adapted to the dynamic needs and goals of users over time.

A DECISION THEORETIC APPROACH
FOR INTERFACE AGENT DEVELOPMENT

I. Introduction

As computers have become commonplace in the Department of Defense, business, and at home, researchers and the software industry have become painfully aware of the need to help users perform their tasks. Determining how to get the right information into the right form with the right tool at the right time is a monumental task. The need exists for solutions to address the problem of increasing task load that is overwhelming the human user. These software solutions could help alleviate user task load by providing abstractions and intelligent assistance that communicates with the user through the existing user interface and adapts its assistance to dynamic user needs and behaviors. The solution should be generic insofar as its benefits can be applied to any highly interactive and information intensive software system. Examples range from virtual battlefield management systems to freight and parcel management systems to Wall Street financial investment and analysis.

To that end, research continues into *interface* or "personal assistant" agents. The purpose of these agents is to reduce information overload by collaborating with the user, performing tasks on the users' behalf (Maes 1994a). Examples of interface agents include office assistance agents, such as e-mail, scheduling, and financial portfolio management agents (Maes 1994a; Sycara et al. 1996; Boone 1998); tutor and coach agents (Chin 1991; Conati, Gertner, VanLehn, and Druzdzel 1997); and character-based assistants for word processors, spreadsheets, and presentation software, such as the Office Assistants found in Microsoft's Office '97 software (Horvitz 1997; Horvitz et al. 1998). Most of these agents have either been pedagogical or

1

narrowly focused (i.e., the agents are useful for a small number of domains and/or model a small fraction of the possible actions).

Reducing user task load involves providing intelligent assistance to the user. Providing intelligent assistance and performing tasks on the user's behalf requires an understanding of the goals the user is performing, the motivation for pursuing those goals, and the actions that can be taken to achieve those goals. The term *user intent* denotes the actions a user intends to perform in pursuit of his/her goal(s). The term *user intent ascription* is the attribution of actions to the goal(s) a user will pursue. That is, user intent ascription is the process of determining which actions are attributable to a specific goal or goals. Therefore, for an interface agent to be able to assist the user in pursuing those goals, the agent must be capable of ascribing user intent to offer timely, beneficial assistance. An accurate user model is considered necessary for effective ascription of user intent.

User modeling is concerned with how to represent the user's knowledge and interaction within a system to adapt the system to the needs of the user. Researchers from the fields of artificial intelligence, human-computer interaction, psychology, education, as well as others have investigated ways to construct, maintain, and exploit user models. The benefit of utilizing a dynamic user model within a system is to allow that system to adapt over time to a specific user's preferences, work flow, goals, disabilities, etc. To realize this benefit, the user model must effectively represent the user's knowledge and intent within the system to accurately predict how to adapt the system. The elicitation, specification, design, and maintenance of an accurate cognitive user model is necessary for effective ascription of user intent.

Current approaches to ascribing user intent (e.g., statistical correlation, AND/OR graphs, rule-based approaches) tend to be reliable only under restricted conditions, to be prone to computational explosion, to use weak updating or learning methods, and/or to fail to be easily extensible. An underlying problem of current interface agent research is the failure to adequately address effective, efficient, and

extensible knowledge representations and associated methodologies suitable for modeling the user's interactions with the system. Current user models lack the representational ability to manage the uncertainty and dynamics involved in ascribing user intent and modeling user behavior for all but limited domains. Ascription of user intent is inherently uncertain. Accurately determining the goal(s) a user is pursuing based on observed actions is a difficult task for several reasons. Users may pursue more than one goal at a time. Actions performed to achieve a goal can be attributed to several goals. Users' needs and goals change over time. Due to the simple fact that most environments where interface agents are utilized are very dynamic, user models must be capable of adapting over time to better model the user.

The integration of interface agents into software applications is eased by the use of an agent development environment and/or tool. Several commercial products exists to perform this task (Mitsubishi Electric Information Technology Center America 1997; Concordia Home Page 1998; IBM 1997; IBM Corp. 1998a; General Magic 1998; Microsoft 1997; ObjectSpace, Inc. 1998), as well as several research agent development environments (Chauhan and Baker 1998; Cohen, Cheyer, Wang, and Baeg 1994; Martin, Cheyer, and Lee 1996; Barbuceanu and Fox 1996a). The underlying problem of agent development is a fundamental lack of methodologies, principles, heuristics, and tools to support the agent development life cycle. To date, no adequate theory of interface agent knowledge engineering and design exists. Every designer of an interface agent reinvents the proverbial wheel. The lack of methodologies for agent-based systems is in no small part attributable to the following two observations: "agent-based engineering" differs significantly from software engineering for direct-manipulation systems (Höök 1997) and artificial intelligence (AI) techniques for agent-based systems differ from "traditional AI" (Maes 1994b). Thus, a foundation on which to base knowledge engineering and design for interface agents will greatly impact the interface agent and user modeling research communities.

3

This dissertation provides such a foundation. The foundation accounts for the need for dynamic, uncertainty-based interface agent user models for user intent ascription. The goal of this dissertation is to advance the state of the art in intent ascription within a software agent paradigm. The dissertation hypothesis can be stated as follows:

> An effective, efficient, and extensible decision-theoretic architecture for user intent ascription improves an interface agent's utility for providing timely, beneficial assistance to the user.

The main thrust of this dissertation is to show that a powerful user intent ascription methodology can be produced by an agent architecture that contains a decision-theoretic user model, combining a Bayesian belief updating reasoning capability coupled with a utility model. The resulting Core Interface Agent architecture is capable of making on-line user model corrections and is shown to be a robust solution to the intent ascription problem. Because the decision-theoretic based strategy is domain-independent, this new methodology for user intent ascription is readily extensible over new application domains. Further, it also offers the ability to "bootstrap" intent understanding without the need for often lengthy and costly knowledge elicitation. Thus, as a side benefit, the process can mitigate the classic knowledge acquisition bottleneck problem.

1.1 Motivation

Several prior publications motivate the work presented in this dissertation. For publications directly resulting from the research presented in this dissertation, see Appendix A. Preliminary research into the use of an interface agent in an expert system shell domain was accomplished (Harrington, Banks, and Santos Jr. 1996a; Brown, Santos Jr., and Banks 1997; Brown, Santos Jr., and Banks 1998a). The resulting architecture, GESIA (Harrington, Banks, and Santos Jr. 1996b), used Bayesian networks (Pearl 1988) as an underlying knowledge representation for user modeling (Brown, Harrington, Santos Jr., and Banks 1997) and reasoning mecha-

nism for predicting user preferences (Harrington and Brown 1997). Usability studies performed using GESIA indicated users welcomed the assistance GESIA provided in helping them do their work (Banks, Harrington, Santos Jr., and Brown 1997). However, GESIA was a simple agent, assisting only when the user first started the system and on very few actions (e.g., tool selection). Furthermore, GESIA's suggestions were incorrect or ignored by users often enough that ways to improve its effectiveness were sought. Key to this improvement is determining what is important to model to make correct suggestions to the user. With that motivation, the work in this dissertation commenced.

1.2 Main Contributions of the Dissertation

A holistic, decision-theoretic methodology explicitly takes into account all aspects of the agent development life cycle, starting with the agent's requirements. Methods for determining when those requirements are not being met and how to correct the agent's "behavior" are developed. For the agent to perform within its environment, one must determine *what is important* to model in the domain, using associated discriminators and metrics to determine and define *when* and *how* to dynamically change the underlying knowledge representation. An explicit representation of users' goals within the domain, with associated metrics to determine *when* a user is pursuing those goals and *how* to offer assistance, allows an agent to ascribe the user's intent. The interface agent uses a decision-theoretic approach to determine *when* and *how* to make suggestions. This approach is paramount to providing timely, beneficial assistance to the user.

This dissertation presents an effective, efficient, and extensible decision-theoretic architecture for user intent ascription. The multi-agent architecture, the Core Interface Agent architecture, provides a dynamic, uncertainty-based knowledge representation for modeling the inherent ambiguity in ascribing user intent. The knowledge representation, a Bayesian network, provides an intuitive, mathe-

5

matically sound way of determining the likelihood a user is pursuing a goal. This likelihood, combined with the utility of offering assistance to the user provides a decision-theoretic approach to offering assistance to the user. The architecture maintains an accurate user model of the user's goals within a target system environment. The maintenance of the user model over time is performed by a collection of correction adaptation agents. The agents base the utility of their individual corrections to adapt the user model on measurable requirements levied on the interface agent.

The main contributions of this dissertation are found in Chapters III, IV, V, and VI and are as follows:

- **User Intent Ascription Philosophy** — Chapter III presents a philosophy of offering assistance to users. The approach is foundational to the construction of a dynamic, uncertainty-based knowledge representation for user intent ascription. This chapter also discusses how interface agents play a key role in symbiotic information retrieval and decision support, a new approach for collaborative, decision-theoretic assistance.

- **Decision-Theoretic Approach** — Chapter IV presents a holistic, decision-theoretic approach to interface agent assistance. The dissertation describes how an interface agent can be used as a decision maker on behalf of the user. The interface agent uses classical decision theory under uncertainty and multi-attribute utility functions, where the attributes are various human factors (e.g., work load, spatial and temporal memory, skill, expertise, etc.). The decision-theoretic approach offers several advantages over current statistical correlation approaches. First, the approach explicitly defines and models those factors affecting the user's decision making process. Purely statistical approaches to offering assistance do not account for the reasons (i.e., why) a user chooses an action in pursuit of a goal. As a result, these statistical correlation methods can only change the probabilities associated with taking an action given an *a priori* situation. A decision-theoretic approach allows the interface agent to

change the underlying reasons. Second, the use of a decision-theoretic approach assures the assistance offered is not only *probably* the correct assistance but also of high *utility* to the user.

- **Core Interface Agent (CIA) Architecture** — Chapter IV also presents the Core Interface Agent (CIA) architecture, which capitalizes on the strengths of the three most prominent research fields in interface agent research: artificial intelligence, human-computer interaction, and user modeling. These strengths, as well as weaknesses, are discussed in Chapter II at a level of detail not typically found in a dissertation. The detailed investigation of the plethora of research into interface agents from the three different fields serves two purposes. First, the investigation provides an insight for researchers from one field into the advantages and disadvantages of the approaches taken in the other two fields. Second, since the research presented in this dissertation draws from all three research fields, the detailed background serves to provide insight into a synergistic approach for interface agent development by reviewing existing ideologies, approaches, techniques, heuristics, methods, and tools used in the three fields.

The CIA architecture provides timely, beneficial assistance to the user, utilizing the decision-theoretic assistance approach. The architecture addresses the need for effective, efficient, and extensible knowledge representations for modeling the user's interactions with the system. The CIA architecture, a multi-agent system, also addresses the need for dynamic, uncertainty-based knowledge representations for modeling user intent. The architecture uses a Bayesian network-based user model that accurately reflects the user's intent. The construction of the user model follows from the user intent ascription philosophy and decision-theoretic approach. The architecture is modularly extensible, providing designers the ability to utilize the architecture in many domains by extending the knowledge captured by the user model.

- **Correction Model** — Chapter V details methods for maintaining an accurate user model integrated into the CIA architecture. Four requirements are levied on the interface agent. The agent's ability to meet the requirements is measured by requirement metrics, which are combined using a multi-attribute requirements utility function. The chapter describes how the CIA architecture can use the resulting requirements utility function to determine how well an interface agent is meeting its requirements. A correction model based on a contractual bidding process, the requirement metrics, and the requirements utility function is presented. The bidding process is begun when the requirements utility functions falls below a threshold. The bidders are a multi-agent system of correction adaptation agents. Each correction adaptation agent possesses a (possibly) unique method for correcting the interface agent, specifically, the interface agent's user model. The correction adaptation agents compete in a bidding process to determine which agent "wins" the bid. A correction adaptation agent wins the bid by providing the greatest improvement to the requirements utility function's value, when evaluated over a history of actions.

The correction model and correction adaptation agents offer several advantages over existing techniques for adapting user models. First, the correction model addresses *when* and *how* to dynamically change the user model and prevents a quixotic search for ways to improve an inaccurate user model. Second, the collection of correction adaptation agents provides many different techniques for improving the user model. Existing user model correction techniques are usually limited to one (machine learning) technique to adapt the user model. Using the requirements utility function, the interface agent can determine the best correction adaptation. Third, the agents are (typically) domain independent and therefore can be used across different domains. This approach supports rapid prototyping of new systems using the CIA architecture. Domain independence is not a requirement of correction adaptation agents. One

8

of the most promising results from this research is that correction adaptation agents can be used for knowledge acquisition. New goals and actions can be added to the user model as the user interacts with his/her environment. This procedural-based approach to user model construction allows the interface agent to dynamically construct the user model without prior domain knowledge. The dynamic construction of the user model virtually eliminates the knowledge acquisition bottleneck that plagues knowledge-based systems. Fourth, this chapter describes a theory for correction adaptation agent evolution, showing how a collection of "atomic" correction adaptations can evolve into fully functional agents. This theory is useful in answering the question "which correction adaptation agents do we build?"

- **Development Environment** — To produce an efficient, effective, and extensible architecture requires an agent development environment that is sensitive to life-cycle changes of the agents composing the architecture. All changes to individual agents or the agent architecture must be managed holistically because required symbiotic reasoning and related information reasoning and decision support is an emergent property of the architecture. That is, the architecture's ability to provide assistance is distributed and the outcome of perhaps many interactions within and among agents. Thus, provisions must be made in an agent development environment to support instantiation and maintenance of this critical functionality, emanating from the collection of agents.

An investigation of tools to support the development of interface agents uncovered major deficiencies in these environments, including a lack of environment specification, adaptivity mechanisms, and knowledge base and reasoning mechanisms. Chapter VI presents a generic interface agent development environment architecture. This chapter describes the details of the architecture, detailing solutions to deficiencies identified in existing agent development environments and how the development environment can support the existing

9

CIA architecture. In particular, the architecture provides a knowledge engineering framework for interface agent development. This framework facilitates the specification and design of interface agents, taking into account compliance with existing computer-based systems and user interface standards. Furthermore, the framework allows for the incremental development of the interface agent user model, to include the Bayesian network-based user model, user profile, and user utility model. The architecture advances the symbiotic information reasoning and decision support goal by providing a comprehensive software engineering, knowledge engineering and acquisition tool.

The multi-agent CIA architecture presented in this dissertation is used in three disparate application domains: a virtual space plane, a probabilistic expert system shell, and a natural language interface information management system. Results indicate the CIA architecture is capable of providing timely, beneficial assistance to users. This assistance is tailored to individual users based on the tasks they are performing, their preferences, biases, abilities, and application usage patterns. Furthermore, the assistance can be adapted to the dynamic needs and goals of users over time.

II. Background

This chapter introduces relevant background on several topics related to this dissertation. The first section discusses interface agents, from the history of interface agent research as it emerged from two contributing research fields (human-computer interaction (HCI) and artificial intelligence (AI)), to agent development environments that can be used to help software developers construct agents. The next section discusses the relatively new research field of user modeling. This discussion shows how user modeling has borrowed from the strengths of the HCI and AI research fields. Throughout the chapter, many examples of systems, including interface agents, utilitizing user models are given. The section proceeds to give a detailed presentation of the elicitation, specification and design, and verification and validation of various user models. Providing assistance to users of a complex system involves being able to decide which goals the users may be pursuing while interacting with the system. Deciding the form of the assistance (i.e., what, when, why, and how) requires decision making involving uncertainty. The last two sections discuss background on both decision theory and a knowledge representation for uncertainty, Bayesian networks.

2.1 Interface Agents

While the term "expert system" was the buzzword of the 1980's, "agent" is the 1990's buzzword. Wooldridge and Jennings (1995), Franklin and Graesser (1996), and Petrie (1996), as well as others, have surveyed the plethora of agent definitions. While there are many types of agents (cf. Franklin and Graesser's agent taxonomy), this dissertation is concerned only with interface agents. For concreteness, the usage of the term "interface agent" is defined, using a hybrid of several definitions found in the literature:

> An adaptive interface agent is a software system situated within a domain environment that adaptively senses, acts, and reacts within this

environment over time, to pursue its own goals and the goal of providing collaborative assistance to the user.

Assistance, as used above, is broadly defined to mean timely, beneficial suggestions, tutoring, help, interface adaptations, etc. for the user. Not all applications are suited for the addition of an interface agent. Waern (1997) states,

> The main application areas for intelligent interfaces are thus such where the knowledge about how to solve a task partially resides with the computer system. Since the user does not know exactly what should be done, he or she cannot manipulate the computer as a tool, but must ask the system to do something for him or her. This request may be incomplete, vague or even incorrect given the user's real needs.

The word "intelligent" is omitted from the definition (and requirements) since an interface agent need not be "intelligent." Waern (1997) requires intelligent interfaces to exhibit some of the requirements this dissertation places on interface agents, including adaptivity (see Section 5.1). Specifically, she states the following techniques connote "intelligence" in interfaces:

- **User Adaptivity** — Techniques that allow the user-system interaction to be adapted to different users and usage situations.

- **User Modeling** — Techniques that allow a system to maintain knowledge about a user.

- **Natural Language Technology** — Techniques that allow a system to interpret or generate natural language utterances, in text or in speech.

- **Dialogue Modeling** — Techniques that allow a system to maintain a natural language dialogue with a user, possibly in conjunction with other interaction means (multimodal dialogue).

- **Explanation Generation** — Techniques that allow a system to explain its results to a user.

Interface agents are a relatively new area of research within the human-computer interface (HCI) and artificial intelligence (AI) communities. Höök (1997)

outlines a number of problems that must yet be solved by these research communities before intelligent user interfaces become a reality. In particular, she states there is a need to develop:

- Usability principles for *intelligent* interfaces versus direct-manipulation systems;

- Reliable and cost-effective intelligent user interface development methods;

- A better understanding of how intelligence can *improve* the interaction;

- Authoring tools that enable easy development and maintenance.

The next two subsections discuss the relationship of interface agents to both the HCI and AI communities in terms of the development history, how both approach the field, and their strengths and weaknesses[1]. While there is no clear delineation between the two communities with regards to their research in interface agents, both approach the research field of interface agents differently. Section 2.3 ends with a discussion on how the field of user modeling has brought the HCI and AI communities together for the purpose representing users' knowledge and interaction within a system.

2.1.1 HCI Historical Development.   One word that characterizes the HCI community's research into interface agents is "customization." The HCI community has concerned itself with what the user can do *with* the agent and the interaction between human and agent. HCI researchers concern themselves with how techniques and methodologies affect *user* performance. The strength of HCI research in interface agents is its attentiveness to the user, focusing on what a user needs to perform his/her tasks, and how best to represent information to the user.

---

[1]While it is not the intention to stereotype a particular research field too narrowly, it is advantageous to point out the general "slant" different research communities take on interface agent research.

Baecker, Grudin, Buxton, and Greenberg (1995) present a history of HCI research[2]. They trace the research from customizable systems to intelligent (interface) agents. Three approaches frame the direction of HCI research:

- Educate developers about users and their work.

- Form active collaborations among users and developers when designing systems.

- Shift responsibility for final design decisions from the initial developers to the users or to people who are closer to the users.

The last approach has been central to the agent research performed by the HCI community. *Customizable* systems allow end users to adjust a system to their specific needs and tasks. These adjustments can be as simple as allowing a user to choose from predetermined alternatives to providing users a way to alter the system itself. Related to this customization is the ability for users to share useful customization with other users.

Several user customization techniques exist, including preferences and templates (cf. Microsoft's Powerpoint templates), customizable workspaces (cf. Microsoft Word's ability to modify tool bars), end-user programming scripts (cf. Novell Groupwise's mail filtering rules), and programming-by-example (cf. any macro recorder). Adaptive interfaces are another customization technique, but one where the user is not in complete control. Adaptations are done either via statistical averages of users or dynamic user models (Baecker, Grudin, Buxton, and Greenberg 1995). Examples of the use of statistical averages include the adaptation of hypermedia menus based on frequency of selection (Akoulchina and Ganascia 1997) and the method used by Harrington and Brown (1997) to select tools, communication modes, and files. Intelligent tutoring and coaching systems are an example of systems with dynamic user models (Conati et al. 1997).

---

[2]Myers (1996) summarizes the (brief) history of HCI technology from the perspective of the impacts of university research on key technologies. Of note, he does not mention interface agents until his final paragraph.

The 1990s saw the emergence of "intelligent" interface agents. Within the realm of HCI research, interface agents are considered "personal assistants" collaborating with the user. Several design considerations are typically considered by the HCI community and include the following (Baecker et al. 1995):

- Give the user a feeling of control.

- Pay attention to the nature of the human-agent interaction.

- Employ built-in safeguards (protect the user).

- Provide the user with accurate expectations.

- Cater to privacy concerns.

- Hide complexity.

HCI has also been concerned with how to represent agents to the human. King and Ohya (1996) investigate the affect of anthropomorphizing (via adding "faces") the interface. Billinghurst and Savage (1996) investigate the use of a rule-based expert system together with voice, gesture, body position and context in a virtual reality interface. Related to this representation are the issues of trust, anthropomorphization, and privacy. Researchers are investigating ways to anthropomorphize agents in an attempt to make the agents more personable and increase trust in the agent's abilities (Koda and Maes 1996). HCI researchers are concerned with the ways to convey the agent's internal state to a user. If the old adage "first impressions are lasting impressions" is taken to heart, HCI researchers are concerned with the user's impressions with the agent. If a user does not trust the agent — typically as a result of failing to understand the agent's "abilities" — future interaction will be affected.

2.1.2 AI Historical Development. A word that characterizes the AI community's research into interface agents is "delegation." In contrast to the HCI community, the AI community as a whole has concerned itself with what an interface agent can do *for* the user. AI addresses how techniques and methodologies affect

agent performance. The community is also concerned with models of agent communication.

Maes (1994b) states that agent research began in 1985, when the term "animat approach" was coined. Maes also does an excellent job of distinguishing "traditional AI" from the study of autonomous agents (what this dissertation terms *agent AI*). In particular, consider the following:

- Traditional AI focuses on "depth" versus "breadth" of knowledge and competence within a domain. Agent AI must exhibit a wide diversity of competence, but at a shallower level.

- Traditional AI focuses on "closed" systems. Agent AI must deal with agents situated *in* the environment, which is typically dynamic in nature.

- Traditional AI deals with one problem at a time; agent AI must contend with multiple, competing problems.

- Traditional AI has static knowledge because it models domain expertise. Agent AI knowledge is dynamic because it models users.

- Traditional AI is not concerned with the developmental aspects of the knowledge structure; agent AI relies on adaptivity of the knowledge structures, incrementally modifying the structures.

Maes discusses the basic problems AI researchers have with adaptive autonomous agents, which includes interface agents. The two basic problems are the following:

- **Action selection** — what to do next given time-varying goals?

- **Learning from experience** — how to improve performance over time?

These problems are far from solved and attempts to address them are active research topics.

In comparison with HCI research for allowing users to explicitly share useful customizations with one another, AI techniques have revolved around agents' communication with one another to share useful information and theories modeling that communication. Architectures and models have been built around agent cooperation and negotiation (Zeng and Sycara 1996; Müller 1996). The term "agency" has evolved out of this area of research. Several AI interface agent projects have successfully married the approach of sharing users' customizations with agent communication (Maes 1994a).

Whereas the HCI approach to interface agent research focuses on a collaboration between human and agent, AI research has relied heavily on the strength of other AI research fields, in particular knowledge representation, machine learning and planning. For example, compare the collaborative nature used by the Agent Building Shell for completing conversation rules (Barbuceanu and Fox 1996b) with Maes' approach to learning new situation-action pairs (Maes 1994a). The former relies on the user to help specify incomplete rules while the latter relies on case-based reasoning to "recognize" new situations.

2.1.3 Interface Agent Examples. In recent years, a plethora of interface agents have emerged. This section is by no means an exhaustive enumeration of interface agents. The interface agents presented were chosen for several reasons. Some were chosen for their historical significance (e.g., they were the first of their kind), while others were chosen for the contribution they made to the field of interface agent research, while yet others were chosen because they represent the "cutting edge" of current interface agent research. This section provides a short background on the types of interface agents and the domains where they have been used to date. Lewis (1998) segregates agents for human-agent interaction (HAI) into three categories: anticipatory, filtering, and semiautonomous.

17

Anticipatory agents attempt to automate some portion of the action-execution cycle. The purpose of these agents is to use information gathering techniques within the confines of the application domain to determine a context used to infer the user's intent. Many of these agents are constructed as adaptive agents, learning a user's preferences, abilities, goals, and needs over time. Two subclasses of anticipatory agents exist: (1) those attempting to learn a user's categorization scheme and (2) those that seek to infer a user's plan of action by observing atomic actions within the domain.

A number of anticipatory interface agents have been integrated into complex application environments. CAP (Calendar APprentice) (Mitchell, Caruana, Freitag, McDermott, and Zabowski 1994) is an agent that learns room scheduling preferences. Users enter and edit meetings on a calendar and instruct the CAP agent to send invitation to the invitees. CAP uses the "looking over the shoulder" approach (Maes and Kozierok 1993), acquiring rules for predicting the duration, location, and times of meetings. Maes and Kozierok (1993) and Kozierok and Maes (1993) present a similar scheduling agent using situation-action pairs to recognize the context of a user's interactions and the appropriate action to take given the situation. COACH is an advisory system for users writing Lisp programs (Selker 1994). COACH builds an adaptive user model of a user's experience and proficiency, and using its domain knowledge of Lisp selects appropriate and timely advice (e.g., a function definition, example, syntax, style guide, etc.) to offer the user. Lumiére is a multi-year research project for integrating intelligent assistance into the Microsoft Office product line (Horvitz et al. 1998). Lumiére uses a Bayesian network user model to determine the probable goals a user is pursuing and the needs associated with the goals to provide assistance to users in meeting those needs. Remembrance Agent (Rhodes 1996) is an automated information retrieval system agent, integrated into the Emacs environment (Stallman 1997). Remembrance Agent uses memory-based reasoning (Stanfill and Waltz 1986) to help users remember relevant information related to their current

goal. Physician's Assistant (Barker 1998), based on IBM's Ginkgo technology (IBM Corp. 1998b), also uses memory-based reasoning. The assistant helps doctors prescribe drugs, including frequency, duration of treatment, and whether it should be taken with meals, for a given situation and recalls a doctor's own practice pattern.

Filtering agents are similar to anticipatory agents in that they also attempt to automate some user actions. Specifically, the filtering agents are concerned with how to automate a user's "filtering mechanism" for determining what is relevant given a scenario and only presenting the relevant information, filtering out the irrelevant information. Examples of filtering agents include the following:

- Maxims (Lashkari, Metral, and Maes 1994), an e-mail filtering agent;

- Ringo (Maes 1994a), a music recommendation system;

- Letizia (Lieberman 1995; Lieberman 1997), one of the first World Wide Web (web for short) "surfing" agents utilizing simple keyword-frequency information retrieval to suggest relevant "near-by" web pages;

- CiteSeer (Bollacker et al. 1998), a web agent for automatic retrieval and identification of relevant publications;

- WebMate (Chen and Sycara 1998), a personal web information retrieval agent utilizing multiple heuristics for determining relevance for further browsing and searching;

- WebAce (Han et al. 1998), a web agent that automatically categorizes and filters a set of documents and performs new queries used to search for relevant related information;

- SHOPPER'S EYE (Fano 1998), a personal digital assistant (PDA)-based agent that uses the concept of physical location-based filtering via global positioning satelites (GPS) to support mall shopping.

Semiautonomous agents are characterized by a user's explicit activation of the agents to perform tasks on the user's behalf. Activation of the agents occurs

via one or more instructional interfaces: form-based interfaces, high-level scripting languages, direct-manipulation interfaces, programming by example, and visual languages.

One of the earliest semiautonomous agents was Information Lens (Malone et al. 1989). The agent's domain is an electronic mail application. Information Lens assists users with the filtering, sorting, prioritization, and general management of e-mail. Users instruct Information Lens via e-mail composed as semistructured messages. The messages are user-defined templates that specify message type (e.g., announcement) with additional header fields (e.g., topic, date). Users import, create, or modify existing active rules for processing e-mail. These active rules serve as simple semiautonomous agents, managing a user's e-mail on their behalf.

One of the most well known software agents employing the semiautonomous notion is the Internet Softbot (Etzioni and Weld 1994). Softbot uses a Unix shell and the World Wide Web to interact with a wide range of Internet resources. For example, the softbot can be "told" to locate all papers that reference this dissertation. The softbot provides an integrated and expressive X-windows, form-based graphical user interface to the Internet, but one which the user explicitly activates. The interface shields users from the underlying subset of first-order logic used as a goal planning language. As a "personal assistant," the softbot is robust in that ambiguity, omission, and errors in user requests are tolerated.

Programming-by-example systems create generalized programs from examples provided by users (Cypher 1991) and bridge the gap between anticipatory and semiautonomous agent systems. Eager is a programming-by-example system for the HyperCard environment (Cypher 1991). It monitors the user's activities and when it detects an iterative pattern, it writes a program to complete the iteration. When Eager detects a repetitive activity, it highlights menus and objects on the screen to indicate what it expects the user to do next. When the user has developed confidence and trust in Eager's suggestions, he/she can instruct Eager to complete the tasks for

him/her. Mondrian, an object-oriented graphical editor that can learn new graphical procedures, is another system utilizing programming by example (Lieberman 1994). A user demonstrates a sequence of graphical editing commands on a concrete example to illustrate how the new procedure should work. An interface agent records the steps of the procedure in a symbolic form, using machine learning techniques, tracking relationships between graphical objects and dependencies among the interface operations. The agent generalizes a program that can then be used on "analogous" examples. The generalization heuristics set it apart from conventional macros that can only repeat an exact sequence of steps. The system represents all operations using pictorial "storyboards" of examples.

2.2 Agent Development Environments

The need for development environment tools to aid designers in the specification and design of agent-based systems is critical. Agent development tools can ease the integration of interface agents into existing environments by supporting the unique requirements of interface agents (e.g., the need for user modeling) and ensuring compliance with existing business practices and user interface standards. However, agent development environments are still relatively immature. One problem lies in the fact that a methodology for domain analysis from the adaptive systems' perspective remains largely a field in its infancy (Benyon and Murray 1993). Furthermore, Sánchez, Azevedo, and Leggett (1995) state "much work is needed to identify an orthogonal set of primitives that would make agent development easier." Additionally, the authors note adaptivity with respects to environmental changes (to include users and their preferences) is a key element that must be addressed. Their project, PARAgent, is investigating these issues, but to date, has not proposed any solutions. Martin, Cheyer, and Lee (1996) enumerate five requirements they feel agent development tools must address. They include:

1. Supporting Conformance — Interoperability is crucial for multi-agent systems. Agents must be designed to interact correctly with other agents. This includes adherence to a communication protocol as well as an agent's "advertised" capabilities.

2. Supporting Heterogeneity — A tool must support a variety of implementation languages, executing platforms, and the mixture between newly developed agents with those adapted from legacy code or information sources.

3. Construction of Agent Communities — The tool must be able to identify sets of agents that can work together to perform a task. The tool should support the ability to determine an agent's capabilities and use (or reuse) them as basic building blocks of a system.

4. Running and Debugging Systems — The tool must also support simulation of system execution. This requirement derives from that fact that many agent-based systems use distributed agents, performing tasks on different machines. The ability to simulate their actions can allow designers to debug agent "behaviors".

5. Facilitating Use of Support Agents — The too, should allow for the customization of specialized agents (e.g., speech recognition or natural language understanding agents) for the system.

It is important to note that the authors' research concentrates on multi-agent systems and not system incorporating interface agents. Therefore, agent development tools meeting their requirements are deficient in several key interface agent requirements, including environment specification, adaptivity mechanisms, and knowledge base and reasoning mechanisms. These deficiencies are discussed in detail in Chapter VI.

Metral (1993) defines eight issues to consider when designing interface agents. The issues are given as follows:

Binding — a means for applications to locate and use agents;

Situation Specification — a means for applications to provide the agent with sufficient state information to make predictions;

Inter-application Object Reference — allowing applications to create objects in the agent's world;

Dynamic Field Negotiation — a means to extend situation specification at run-time;

Action Specification — a means for the agent to specify which action the application is to execute and on what objects it is to be executed;

Situation-Action Matching — matching user or agent predicted actions to unresolved situations(s);

User Feedback — a specification detailing how to provide real-time feedback to the user about status of the agent, to include explanations of actions.

This section discusses only development environments. There are other methodologies for developing agents. For example, there exists a number of agent programming languages (e.g., AOP (Shoham 1993)) and agent communication languages for developing agents (e.g., COOL (Barbuceanu and Fox 1996b), KIF (Genesereth and Fikes 1997), KQML (Mayfield, Labrou, and Finin 1996), D'Agent (formerly known as Agent Tcl) (Gray 1995a; Gray 1995b; Gray et al. 1997)). Several agent frameworks and tools have used Java (Sun Microsystems 1998) to implement mobile agents system, including IBM Aglets (IBM Corp. 1998a), Concordia (Mitsubishi Electric Information Technology Center America 1997; Concordia Home Page 1998), JAFMAS (Chauhan and Baker 1998), JATLite (Stanford University Center for Design Research 1998), Odyssey (General Magic 1998), and Voyager (ObjectSpace, Inc. 1998). These Java-based tools focus mainly on building secure network-based applications that use mobile agents to search for, access, and manage corporate data and other information. They provide an application programming interface (API) to Java classes and/or agent managers (e.g., agent name

servers). However, the development environment tools presented below envelop and subsume many of these other methodologies and therefore, this section concentrates on the development environments alone, mentioning the subsumed methodologies as appropriate. As an analogy, we are not interested in C++ libraries, but in development environments that use C++ and provide tools to use it (e.g., Microsoft's Visual C++). Furthermore, the examples concentrate on those environments suited for interface agents.

In the following subsections, the most promising agent development environments for human-agent interaction are discussed in detail. As will be discussed later, the agent development environment problem is still an active research area. Several of the discussed environments provide a visual development environment, with associated methods for specifying agent "behaviors" via an agent communication language (Cohen et al. 1994; Martin et al. 1996; Barbuceanu and Fox 1996a; Chauhan and Baker 1998; Coen 1994; IBM 1997; Reticular Systems 1998). Most of the projects (commercial and research) provide a simple syntactic check on the agent's specification.

2.2.1 Agent Building Environment. IBM's Agent Building Environment (ABE) is a software developer's toolkit that eases integration of agents into legacy software (IBM 1997). IBM's toolkit provides sensing "adapters" and simple *if–then* rules to control the overall actions of the agent. Agents perform their task(s) if the antecedent of their "activation" rule is satisfied. The knowledge base is stored in a library for use by all agents.

2.2.2 Agent Building Shell. Barbuceanu and Fox (1996a) address issues related to actual programming constructs for internal representation and reasoning of agents and the development of tools to support how an agent represents its "view" of the world, to include updating this representation based on interaction within the environment. The Agent Building Shell (ABS) (Barbuceanu and Fox 1996a) con-

24

tribution to the field of agent development environments is the attention given to an agent's internal knowledge representation and reasoning about the environment. The authors use rule-based descriptions, called conversation plans, of agent actions in a given situation. A conversation plan specifies the available conversation rules, their control mechanism and the local data-base that maintains the current state of the conversation. Conversations can be represented graphically with finite state automata. Error recovery rules can be specified to handle incompatibilities between the current state and incoming messages and are invoked when a conversation rule cannot handle the current situation. Rules are specified with a coordination language. The authors assume a deterministic environment. For their environment, this assumption is reasonable. However, for interface agent development, where the domain (including the user's behavior within the domain) is not deterministic, developers must chose knowledge representations capable of representing uncertainty. ABS rules must also be specified by the user.

2.2.3 AgentBuilder. AgentBuilder is a commercial integrated agent development tool suite (Reticular Systems 1998). The suite has two main components, the Toolkit and the Run-Time System. The AgentBuilder Toolkit includes tools for managing the agent-based software development process (Project Control tools), analyzing the target system and agent domain (Ontology Manager), designing and developing networks of communicating agents (Agency Manager), defining behaviors of individual agents (Agent Manager) to include learning and planning capabilities, and debugging and testing software (Agent Debugger). The Run-Time System includes an agent engine that provides an environment for execution of the developed agent software. Both the Toolkit and Run-Time System are implemented in Java. AgentBuilder utilizes KQML performatives as the agent communication language. The standard set of KQML performatives can be extended with AgentBuilder.

25

2.2.4 Open Agent Architecture. The goal of the Open Agent Architecture project (Cohen et al. 1994) is to develop an open (e.g., extensible) agent architecture and accompanying user interface for networked desktop and hand-held machines. This system supports multiple agents for distribution of users' tasks and inter-operability between application subsystems. Martin, Cheyer, and Lee (1996) describe the Agent Development Tools (ADT) that allow users to generate code "stubs" for several popular programming languages. Their system includes two other tools — the Linguistic Expertise Acquisition Program (LEAP), for interfacing a new agent with existing linguistic support agents such as natural language parsers and speech recognition systems, and PROJECT, for creating and maintaining repositories of reusable agents.

The Open Agent Architecture's Agent Development Tools allow designers to specifying the agents' interfaces in an extension of Prolog. Agents define their capabilities (called *solvables*) and inform a facilitator agent of those capabilities. The facilitator is responsible for delegating tasks based on the capabilities. The use of a Prolog-like agent communication language allows the facilitator to reason over the capabilities of the agents, i.e., determine if its invocation conditions are true and what the needed parameters for invocation are. This reasoning is only about the agent's explicitly defined interface, and does not consider its internal capabilities unless these internal capabilities are explicitly defined as an external capability.

2.2.5 SodaBot. The philosophy behind the SodaBot research project (Coen 1994) is that software agents should be written using a vocabulary not provided by traditional programming languages — it should be possible to create agents solely by specifying their abstract behavior. SodaBot is a general-purpose software agent user-environment and construction system. Its primary component is the basic software agent — a computational framework for building agents which is essentially an agent operating system. The SodaBot project also presents a new language for programming the basic software agent whose primitives are designed around human-

level descriptions of agent activity. Via this programming language, users can easily implement a wide-range of typical software agent applications, e.g., personal on-line assistants and meeting scheduling agents.

SodaBot provides the following tools:

- A universal computational framework — the Basic Software Agent — for creating and using agent applications;

- A very high level agent programming language — SodaBotL – that provides the right level of abstraction to allow simple construction of complex agent applications. SodaBotL is used to program the basic software agent;

- A graphical user-interface;

- Automatic distribution of software agents across the Internet.

SodaBot has recently been replaced by Metaglue (Michael H. Coen (mh-coen@ai.mit.edu) 1998). Details on this new architecture are not yet published.

2.3 User Modeling

User modeling, as a research discipline, emerged in the 1980s, from research being done in the artificial intelligence, human-computer interaction, psychology, education, and other research fields. In the previous sections, we concentrated on the first two research fields — HCI and AI — discussing their impact on the development of interface agents and their strengths and weaknesses. In this section, we discuss how the field of user modeling has brought the HCI and AI research communities together for the purpose of representing users' knowledge and interaction within a system and a discussion on user intent and its applicability to user modeling. Several of the ideas presented in this section, as well as the general flow of the presentation, follow Loren Terveen's article (Terveen 1995) "Overview of Human-Computer Collaboration."

27

2.3.1 User Modeling Historical Development. The AI community is concerned with the internal knowledge representation of interface agents versus the HCI community's concern with representation of the agent to the user. The field of user modeling is perhaps a good marriage between artificial intelligence and human-computer interaction. User modeling is concerned with how to represent the user's knowledge and interaction within a system to adapt those systems to the needs of users. Benyon and Murray (1993) state that user models are one of three models[3] all adaptive systems must possess. The main purpose of user modeling is to determine what the user intends to do within a system's environment for the purpose of assisting the user. Ntuen (1997) defines human (user) intent as "mental states which drive actions." As previously stated, the term "user intent" as used in this dissertation is used to denote the actions a user intends to perform in pursuit of his/her goal.

The infusion of ideas from many disparate research fields has allowed user modeling researchers to benefit from the important contributions of each of the separate contributing research fields. For example, the user modeling community has been able to reap the benefits provided by artificial intelligence researchers by various knowledge representations developed by AI researchers. The methods used include logic-based techniques (Pohl and Höhle 1997; Giangrandi and Tasso 1997), abductive reasoning-based techniques (Csinger and Poole 1996; Csinger et al. 1994), machine learning techniques (Chiu et al. 1997; Doux et al. 1997; Paranagama et al. 1997; Gori et al. 1997; Ambrosini et al. 1997), Bayesian methods (Conati et al. 1997; Brown et al. 1998; Harrington and Brown 1997; Horvitz et al. 1998; Albrecht et al. 1997; Noh and Gmytrasiewicz 1997; Corbett and Bhatnagar 1997), and neural networks (Paranagama et al. 1997; Gori et al. 1997; Ambrosini et al. 1997). The human-computer interaction research impact on user modeling can be seen in the use of user models to customize presentation of information (Gutkauf

---

[3]Benyon and Murray include the user model, the domain model (a model of the system), and an interaction model (a model of the user-system interaction).

et al. 1997; Hirst et al. 1997; Kalyuga et al. 1997), to provide feedback to users about their knowledge in a domain (Bull 1997), and to help users locate useful information (Linden et al. 1997; Maglio and Barrett 1997; Akoulchina and Ganascia 1997). Additionally, user models have taken into account various human factors, such as a user's psychological ability, working memory, task load or cognitive load, to adapt a user interface and/or the information presented to the user (Mulgund and Zacharias 1996; Schäfer and Weyrath 1997; Stefanuk 1997; Gavrilova and Voinov 1997; Banks, Stytz, Santos Jr., and Brown 1997; Brown, Santos Jr., and Banks 1998b; Keates and Robinson 1997) The use of user models has been shown to increase effectiveness and/or usability of systems implementing the various techniques from the field of user modeling (Jameson, Paris, and Tasso 1997).

Vassileva (1997) believes researchers need to start using lessons learned from user modeling to impact the way they view interactive human-computer environments. She discusses viewing these environments along three orthogonal dimensions: elements — the goals, plans, resources, and actions composing the atomic entities an agent (human or otherwise) is concerned with; processes — the types of processing (e.g., reaction, deciding, learning) that takes place in an agent; and relationships — the way agents interact with one another. Her approach makes explicit the reasoning about the purpose of adaptations, treats human and computer agents the same in the environment, takes into account user motivation, emotions, and moods, and presents a unified model of collaborative, cooperative, and adverse behavior. The research presented in this dissertation addresses many of her concerns.

2.3.2 Elicitation of User Models. Elicitation of user models is a knowledge acquisition process. It is well-known in the artificial intelligence research community that knowledge acquisition is the bottle-neck of intelligent system design. However, unlike knowledge acquired for an AI system such as expert systems, the "life" of a user model may be short-lived. Some user models are only useful for the duration of a problem solving "exercise" (e.g., student modeling and/or decision support user

models (Conati et al. 1997)). The purpose of the user modeling component in an intelligent system has a great affect on the elicitation techniques used to construct the user model.

Knowledge in a user model may be acquired either implicitly via inferences made about a user, or explicitly via a collaborative process with the user, or a combination of both. Both acquisition approaches may be done before using the system, "on-line" while using the system, or "off-line" after using the system. Determining when and how to elicit the user model knowledge is a domain and application dependent decision. For example, an application that will be used by a small group of users with well-defined input and output (e.g., accounting applications) may use a user model declaratively defined during the design of the application. Knowledge could be acquired using standard knowledge elicitation techniques (Cooke 1994). On the other end of the spectrum, an application used by a diverse group of users (e.g., web based applications, word processors, decision support systems) with ill-defined and/or dynamic input and output must be capable of adapting to the needs of the user and the task at hand. Therefore, the user modeling component in these applications must be dynamic, adapting as the user interacts with the application. The next section discusses several different machine learning techniques with respect to their usefulness in various domains.

2.3.2.1 Machine Learning Techniques for Elicitation. Horvitz et al. (1998) explains how detailed (and time consuming) user studies can be used to better understand the needs and behavior of users as they encounter problems using software applications. This understanding can then be translated into knowledge for use in a user model. Unfortunately, not all user model designers have the resources to be able to elicit user models by watching users perform their work. Machine learning techniques are the most common and widely used methods of eliciting user models implicitly from users. The main reason for the popularity of machine learning techniques for user model elicitation is the fact that the user model may be elicited

30

incrementally, typically without user intervention by watching users performing their tasks. Therefore, machine learning techniques attempt to avoid the bottleneck of knowledge acquisition for user models.

Perhaps one of the most popular heuristics for elicitation is statistical correlation. That is, actions a user performs more frequently, given a "situation," are good candidates to try again given the same situation. Due to its popularity in the research, the next several paragraphs discuss several key pieces of research related to statistical correlation.

Maes uses memory-based reasoning (Stanfill and Waltz 1986) to learn *situation-action* pairs in an electronic mail application, meeting scheduler, and Web news reader domain (Maes 1994a). Memory-based reasoning is a frequency-based machine learning technique. Situations are composed of vectors of features relevant to the environment the system is situated within. For the e-mail application, this might include the sender and recipient of a message, "Subject:" line keywords, the message length, the message urgency, etc. The relevant features to be included in a situation are typically determined during the system design phase. As the user interacts with the application, situations (defined by the features) and the actions taken by the user for a given situation are recorded. An *interface agent* uses the raw situation-action pair data to attempt to offer assistance to the user based on a closest match to previous situations. The frequency of the situation-action pairs determines their applicability. The statistical correlations are updated off-line. Maes uses two thresholds to determine when the statistical correlation is "strong" enough to warrant interface agent action on behalf of the user.

Bauer investigates the acquisition of user preferences for plan recognition, in a Web news reader domain (Bauer 1996). He uses an off-line ID3 decision tree inductive learning algorithm (Quinlan 1986) to learn classes of situations based on user actions. Bauer states his approach is better than the approach used by Maes. In particular, memory-based techniques maintain only the "raw data" in a situation-action

31

database, while inductive techniques attempt to draw higher level abstractions from the situations. Additionally, situation-action pairs are restricted to one-step predictions of single user actions, preventing the direct application of situation-action pairs for plan recognition, where the evidence has accumulated over several observation steps.

Several other situation-action pair deficiencies exist. First, determining the relevant features to model the situations is critical. Within a restricted domain such as news readers, this is a tractable problem. However, for many domains, this restriction makes the situation-action pair technique inadequate. This problem is not unique to situation-action pairs[4]. Furthermore, to match an action to a situation, the situation must be fully specified. In memory-based reasoning, each feature in the situation vector is relevant by default. User model designers must be able to specify the situation unambiguously. Due to inherent uncertainty in many environments, due to faulty sensors, unobservable factors, etc., designers may not be able to specify a situation vector unambiguously. Second, the user must perform a number of actions prior to making any decisions. This problem can be overcome by using additional elicitation techniques such as user profiles or stereotypes providing *a priori* knowledge about a user. Third, most current situation-action pair-based user models ignore the *utility* of offering assistance to a user, assuming frequency is *the* key determining factor of user behavior. Systems must not only be capable of determining the frequency of an action given a situation, but the relevancy based on many factors, to include the goals the user is pursuing, cognitive factors such as spatial and temporal ability, skills proficiency, etc. "Situations" typically do not capture such factors. Fourth, situation-action pairs ignore the correlation between actions (i.e., behaviors) and the goals being pursued. Situations are matched with a single action to perform, given the situation. However, in many situations, a user may perform a

---

[4]Bauer states "the question of how to come up with an appropriate feature set for the description of situations can be crucial."

series of actions to accomplish some higher level goal. Simple situation-action pairs cannot model situation-goal-actions. Lastly, while most user models using situation-action pairs allow for machine learning techniques to extend the behavioral model of users by adding new pairs and/or updating the statistical frequencies, they fail to capture the inherent uncertainty and dynamics of predicting the user's intent.

2.3.3 Specification and Design of User Models. Determining how to specify and design a user model to accurately model a user is difficult at best. The work done in artificial intelligence knowledge representation research is of particular use here. Many research interfaces use rule-based intelligence (Thomas 1993). Rule-based representations, like those used in most intelligent user interfaces, fail in two key areas: representing uncertainty and dynamic user modeling. The use of "probability modules" (Winston 1984) is an *ad hoc* approach to determining answer reliability, i.e., uncertainty. Furthermore, the addition and deletion of rules to dynamically model a user is *ad hoc*. Therefore, knowledge representations that can dynamically capture and model uncertainty can improve the user modeling in an intelligent user interface. In recent years, numerical uncertainty techniques from the AI community have been used in a number of systems to capture the uncertainty inherent in modeling users (Jameson 1996).

In general, two main schools of thought exist on the design of user models. The first uses "hand-coded" user models. That is, the system designer determines how best to model the users *a priori*. (See Jameson (1996) and Maes (1994b) for examples). Hand-coded user models are typically static. Once they are designed, they will not change structure. The second method uses "machine-coded" user models. That is, a user model is constructed by the system as it "learns" more about the user. These models are dynamic (i.e., the structure and/or "contents" of the user model changes over time). (See Jameson (1996) and Harrington and Brown (1997) for examples.) Both methods have advantages and disadvantages and the particular domain will determine which method best meets the needs of the user model's con-

struction. It should be obvious that the elicitation method(s) used may be integrally tied to the way system designers specify and design the user model. This dissertation is concerned with other issues that impact both hand and machine-coded user models discussed below.

2.3.3.1 Relevancy and User Intent.   As mentioned previously, to accurately predict user intent, an interface agent must have an accurate cognitive model (i.e., user model). Modeling every possible naturalistic property in the user's world fortunately does not lead to the most accurate model (DeWitt 1995). If this were not the case, there would be little hope in using numerical uncertainty management techniques such as Bayesian networks due to the computational inefficiency of large networks (Cooper 1990).

DeWitt notes that not all naturalistic (i.e., observable) properties play an interesting role in a user's causal model (DeWitt 1995). That is, only certain observable actions and information in a user's "world" will have relevance to that user. DeWitt uses the term *causally efficacious* to describe naturalistic properties that "play any interesting causal role in cognitive functions." He argues that not everything observable is of interest when we make decisions. DeWitt's philosophical argument has direct analogy in user models. A user model must not include every possible piece of information about "the world." To use DeWitt's example, while the manufacturer of a stereo, the amount of dust on top of the stereo, and the weight of the unit are all observable properties, none of this evidence would likely have any bearing on the reasoning as to why the stereo does not work. To include such information in a user model needlessly complicates the model, not only semantically, but computationally. To take this analogy one step further, the less causally efficacious a property is to another, the more likely it can be ignored. Therefore, to effectively and efficiently capture user intent, the user model should not attempt to model every possible action the user may exhibit, but only those that are relevant.

34

Harrington and Brown (1997) use the term *relevancy set* to describe those properties included in their user model, represented by an *interface learning network*. The interface learning network is a Bayesian network supplemented by information support nodes storing the statistical frequency of user actions. For hand-coded models, the relevancy set will not change. For machine-coded models, the relevancy set may change with use. A *relevancy neighborhood* are those properties that are immediately causally efficacious to the decision under consideration. As an example from Harrington and Brown, a user model may contain the possible communication modes and tools a user may use in a system, given the user's class and individual preferences learned by the user's interface learning network, as well as several knowledge bases used previously. These nodes in the network are considered the relevancy set. When the interface learning agent wants to predict if a particular knowledge base will be needed by the user, the other knowledge bases are in the relevancy neighborhood. Therefore, the relevancy presents a computationally efficient and semantically meaningful view of the user's relevancy set at any given time. The concept of a relevancy set is a good tradeoff between computational complexity and representational exactness, while maintaining full semantics.

2.3.3.2 Specification and Design Examples. Various types of models of the user can be elicited. Stereotypes (Rich 1983) represent "typical" users, expressing various traits, characteristics, and attributes that are common among a group of users. Users classify themselves as belonging to a particular stereotype. User profiles are used to represent background, interests, and general, typically static, knowledge about a specific user. User profiles may be elicited from users by, for example, standardized tests, surveys, and/or assessments. These elicitation techniques are used to construct the user profile.

User modeling researchers, as well as other research disciplines, have used stereotypes and user profiles for various purposes (Kay 1994). User modeling examples include using stereotypes to filter World Wide Web documents (HTML/text),

basing a user's inferred interests on user adapted stereotypes acquired via experts (Ambrosini, Cirillo, and Micarelli 1997), to derive initial proficiency estimates on a user's level of advancement for use in computer-assisted language learning (Murphy and McTear 1997), and adapting Web-based hyper-media based on stereotypical users' abilities (e.g., disabled and the elderly) (Fink, Kobsa, and Nill 1997).

As an extended example, Csinger and Poole (1996), as well as Csinger, Booth, and Poole (1994), use a simple form-filling operation to elicit interest metrics for "intent-based" authoring. Simple Horn-clause dialect (Poole 1993) is used to represent the contents of the knowledge-bases that compose their user model. Use of Horn-clauses allows inferencing to be performed using a best-first probabilistic Horn-clause assumption based system. The user model is represented by a set of recognition assumptions that satisfy the observations of the user's actions. The abductive reasoning engine is used not only to determine the most plausible user model, but the best (i.e., least costly) presentation of the information in the system.

As mentioned previously, modeling users involves a great deal of uncertainty. Using knowledge representations capable of dealing with this uncertainty in a sound, mathematical way is desirable. Jameson (1996) provides an excellent overview of the use of numerical uncertainty techniques in systems utilizing user modeling systems. In particular, he focuses on Bayesian networks (Pearl 1988), Dempster-Shafer Theory, and fuzzy logic knowledge representations. In addition to categorizing each system by its knowledge representation, he also categorizes each system by its input (termed observable states and events), long- and short-term cognitive states, and domain. Furthermore, he critically discusses the advantages and disadvantages of each knowledge representation from the viewpoints of knowledge engineering requirements, programming effort, empirical model adjustment, computational complexity, "human-likeness," justifiability, and explainability.

One of the most widely used applications employing an uncertainty knowledge representation user model is the Microsoft Office '97 applications. Horvitz et al.

(1998) present their work on the Lumiére project. The Lumiére system utilizes a Bayesian network user model to capture the needs and goals of users within the domain of the Microsoft Excel spreadsheet application. The Bayesian network is elicited from application designer experts who had the advantage of performing "Wizard of Oz" studies[5] on real world users. To aid designers, a Lumiére Events Language was developed that allowed atomic application events to be directly modeled, as well as streams of atomic events to be formed into Boolean and set-theoretic combinations of low-level events. Furthermore, the language allows designers to compose new modeled events from previously defined modeled events and atomic events. The language allowed the researchers to build and modify transformation functions that would be compiled as run-time filters for modeled events. Horvitz et al. discuss the framing, constructing, and assessing of Bayesian user models. The "Wizard of Oz" studies revealed classes of evidential distinctions, providing observational clues valuable to making inferences about a user's problems and the user's need for assistance. The identified classes are as follows:

- **Search** — Repetitive, scanning patterns associated with attempts to search for or access an item or functionality.

- **Focus of attention** — Selection and/or dwelling on artifacts in the interface.

- **Introspection** — A sudden pause after a period of activity, or significant slowing of activity.

- **Undesired effects** — Attempts to return to a prior state. This includes undoing actions, or closing a dialog box shortly after it is opened without invoking an offered operation.

- **Inefficient command sequences** — Performing operations that could be done more simply or efficiently.

---

[5]"Wizard of Oz" studies involve users interacting with applications (e.g., Microsoft's Excel) while a human expert provides application assistance to the users. The users are unaware the assistance is being provided by human experts. The effect is a person "behind the curtain."

- **Domain-specific syntactic and semantic content** — Consideration of special distinctions in content or structure of documents and how a user interacts with these features.

Structuring effective Bayesian user models relies on the ability to define appropriate variables and states of variables in the Bayesian network and to assess probabilities over these variables. The structure of the Bayesian network must also take into account temporal reasoning about a user's actions. Horvitz et al. investigated the use of dynamic Bayesian networks and single-stage formulations of the temporal variables. Dynamic Bayesian networks consider dependencies between variables within as well as between time slices (Nicholson and Brady 1994; Albrecht, Zukerman, Nicholson, and Bud 1997; Schäfer and Weyrath 1997). Single-stage formulations determine the likelihood of alternative goals at the present moment and embed considerations of time within the definitions of observational variables (e.g., user performed action x less than y seconds ago) or introduce time-dependent conditional probabilities.

Karagiannidis, Koumpis, and Stephanidis (1996) present an approach to determine, within the domain of intelligent multimedia presentation systems (IMMPS), what, when, why, and how, to adapt the system's presentation. Central to their approach is an explicit decomposition of the adaptation process into adaptivity constituents (the "what"), determinants (the "when"), goals (the "why"), and rules (the "how"). Their approach does not address agent-based environments, although the authors are outlining a project concerning the implementation of their approach within an agent-based environment[6].

The authors' work has several weaknesses. Karagiannidis et al. have no way of determining whether their method of adaptation, the "how," is feasible within their approach, nor its impact. The authors' approach is to rely on the application

---

[6]Personal communication with Constantine Stephanidis.

to determine whether the adaptation method is feasible[7]. This places an unnecessary burden on the application designer to account for this. Furthermore, it makes integration of agents into legacy software nearly impossible. From a computational stand-point, certain adaptations could be abandoned given the evidence that the approach would not be feasible. All goals within a system deal with adaptation of the presentation. These are not necessarily explicit user goals. Therefore, the assistance offered may not help the user achieve a goal they are pursuing directly, but may indirectly help them by presenting the information in the "best" (as determined by the designer) way.

Yoshida (1997) uses a directed graph to represent the user's use of files and other system resources in the UNIX operating system[8]. The author states that user models acquired by automating a user's sequence of commands do not always typify the user's behavior. Yoshida's work investigates the importance of data dependency between commands the user invokes. Experiments show that taking into account data dependency yields a better user model and improves command selection accuracy. Yoshida's framework involves using the directed graph to model relationships between the commands a user invokes and the files needed by those commands. The graph representation is particularly well-suited to multi-tasking environments such as UNIX. As the user interacts with the operating system, the commands and input/output access is "recorded" in the directed graph. This graph is then analyzed via graph-based induction. Typical subgraphs are extracted from the input graph so that the subgraphs represent typical events in the system. These extracted subgraphs are the user models and are used to pre-fetch needed files based on prediction of future commands of the user.

Ntuen (1997) uses the concepts of finite-state grammars to generate command languages. The author states that command languages have some characteristics of

---

[7]Personal communication with Constantine Stephanidis.
[8]For a more detailed presentation, see Yoshida and Motoda (1997).

certain finite-state grammars implying command languages can be formulated and represented by production rules generated by a finite state grammar. The author also states that communicated intents, driven by performing activities, have properties similar to command languages. The usefulness of this representation comes in the prediction and adaptation to users by representing their intents in a knowledge base and inferencing over this knowledge base to determine what the user is attempting to do. The knowledge representation works well in application domains where there is little ambiguity concerning what a particular activity might convey in the way of intent; i.e., domains where tasks are well defined.

Stary (1997) discusses how designers can specify and design intelligent user interfaces through the identification of human task and work flows. Stary presents the TADEUS (Task Analysis, Design, End-User Systems) approach, a task-oriented system development approach. His approach deals with designing intelligence interfaces by addressing what a user needs for task accomplishment. This is in contrast with using machine learning techniques to elicit "intelligence" although his approach does not preclude using machine learning techniques. Using the TADEUS approach, work flows are migrated into the user-interface design representations. Business goals and rules and their relationships to tasks and people involved can be integrated into the user interface design. Meaningful sequential activities for task accomplishment are determined. Profiles of various users of the system, and their functional roles to the task and work flows can be generated. Flow and control of data can be made transparent and interface designers can provide notation that allows static and dynamic specification and adaptation of the work flow models. The TADEUS approach allows consistent and context-sensitive user interface development to be supported in the software development process and eases the use of artifacts of previous development.

2.3.4 User Model Verification and Validation. Once the user model has been elicited, specified, and finally designed, determining if the user model is an accurate representation of the user's actual interaction with the system and meets

40

all designer requirements is paramount to the functionality of the user model in a system. The accuracy of the user model must be measured if we are to determine how closely the user model matches the real (exhibited) behavior of the user. Since the user's behavior may change over time, deviating from the the originally elicited, specified, and designed user model (assuming we accurately captured the user's model initially), verification and validation techniques must be used *over time* to make sure the user model currently reflects the user's behavior.

Note that user model verification and validation is different than the use of machine-learning techniques to adapt the user model. User model validation is concerned with determining if we built the right user model, while verification is concerned with determining if we built the user model correctly based on the specification. Most machine-learning techniques used in user model adaptation fail to measure how closely the user model reflects the user's behavior. Typically, usability studies are performed for this purpose. Users are asked a number of subjective questions concerning their likes and dislikes of the system, and from these questions, user model and/or system designers ascertain the validity of the user model. Most usability studies are done "off-line" and have no immediate bearing on the user model. Objective metrics are useful for *dynamically* modifying the user model to better model the user. Currently, this is an fertile area of research.

How to change a user model is just as important, if not more so, than when to change the model. Is it better to fail to recognize a user model is no longer adequate than to change a user model incorrectly? Both problems have the same effect — an incorrect user model.

2.3.5 Utilization of User Models. In practice, user models used in "real" systems make several assumptions, as described by R. V. London[9]:

---

[9]As cited by Järvnin (Järvnin 1993).

- **Closed-world assumption** — All relevant plans and actions are known either explicitly or by closure.

- **User's correctness** — The user has a coherent, well-formed plan, with no fatal misconceptions.

- **Unified goal and plan** — Typically, the user has a single top-level goal, with subgoals in a proper hierarchy. If there are non-hierarchic goals, then they are either resolved into a unified plan or maintained as multiple plans that are nearly independent. Additionally, users have a somewhat persistent focus.

- **Co-operative user** — The user is purposely giving information to help the user modeler, or the system can verify and adjust its conclusions by cooperative negotiations with the user.

- **No real-time requirements** — The system is not required to respond within the time bounds of normal human communication.

Pohl (1997) further discusses shortcomings of user models utilized in most systems:

- Assumptions about interaction preferences or behavior patterns are missing.

- Assumptions are acquired with specialized heuristics, drawing from isolated observations without regard to interaction context.

- User behavior, preferences, and mental attitudes are subject to change, but not adequately modeled.

- User models are constructed and exploited mostly within the limits of one application. However, it is beneficial to share information about users among several applications.

2.3.6 Plan Recognition. Plan recognition is the task of ascribing intentions about plans to an agent (human or software), based on observation of the agent's actions. There are three types of plan recognition: intended — the agent chooses

actions that make the plan recognition process easier; keyhole — the agent is unaware of or indifferent to the plan recognition process; and obstructed — the agent is aware of and actively obstructs the plan recognition process (Waern 1996).

Several researchers have used plan recognition as a user modeling technique. Plan recognition has been used to predict actions in a virtual predator-prey environment (Foner 1994) and to predict users' actions in games (Albrecht et al. 1997). Several authors have investigated the use of probabilistic approaches, including Bayesian networks (see Section 2.4) for plan recognition and generation.

Kirman, Nicholson, Lejter, Dean, and Santos Jr. (1993) investigate the use of a Bayesian network of the "world" — in the authors' case, a simple room with a mobile robot and a target to be found — and a utility measure on world states to generate plans (sequences of actions) with high expected utility. The number of plans investigated are restricted to those with high utility with respect to attaining a goal. In the authors' work, the goals are known with certainty and the plans to achieve the goals must be found. They show that by using decision-theoretic methods, they can drastically reduce the number of plans that must be investigated.

Waern uses keyhole plan recognition to determine what a user is doing within a route guidance domain and Internet news reader (Waern 1996). She uses pre-compiled plans called *recipes*. Assistance is offered for attaining a goal based on the calculated probability the user is pursuing the goal. Her approach does *not* address the utility of offering assistance. She considers only the probability that offering to perform an action will help the user obtain a goal. Furthermore, her approach is not dynamic; that is, there is no adaptation of the pre-compiled plans. Lastly, there is no way to *trace* the user's execution of a plan nor offer explanations of assistance.

ANDES is a system that performs long-term knowledge-assessment, plan recognition, and prediction of students' action during physics problem solving (Conati, Gertner, VanLehn, and Druzdzel 1997; Gertner, Conati, and VanLehn 1998). ANDES uses Bayesian networks constructed dynamically from static knowledge bases

of specific physics problems as well as physics concepts, laws, formulas, etc. The plan recognition component is used to tailor support for the students when they reach impasses in their problem solving process. Their approach uses explicitly stated goal and problem solving strategy nodes to model the user's problem solving, but does *not* use a utility-based approach. The authors note they are researching methods to expand the fixed knowledge base used to construct the Bayesian networks; in particular, they are adding knowledge to determine common "misconceptions" within the domain and changing the static parameters.

The Pilot's Associate Program was a research effort using plan recognition within a real-time domain to determine goals from actions (Banks and Lizza 1991; Small 1995). The Pilot's Associate was a software agent providing assistance — in the form of wanted and needed information at the correct time — to a combat pilot. The Pilot's Associate pushed the envelope in knowledge representation and reasoning techniques. They used AND/OR plan-and-goal trees with an associated "dictionary form" for explanations of plans a user was pursuing, and offered assistance to help the pilot. The graph's hierarchical structure produces a common planning language for use among heterogeneous modules. One of the main problems with their approach was due to the technology available at the time for representing and reasoning about uncertainty. As a result, their approach did not account for the uncertainty nor utility in obtaining goals by performing actions.

Jameson (1996) presents an overview of a casual planning model that can be summarized in terms of the following phases:

1. A user observes aspects of the physical environment (e.g., stimuli).

2. The user compares the observations with his/her goals, determining the extent to which her goals are fulfilled.

3. The user selects a plan to address an unfulfilled goal.

4. The user refines a plan, taking into account the current situation.

5. The user executes a sequence of actions.

6. These actions have observable effects on the environment.

Plan recognition is made difficult because many times users do not follow pre-planned goals. They perform some actions that can be ascribed to more than one plan, and may be pursuing several plans at once. Because of this, ascribing intentions to a user's actions by observing the actions to those actions can be next to impossible. One way of avoiding this is to observe only the most recent actions (Waern 1996; Foner 1994). A *fading function* is used to "forget" past actions. Not only does this have the advantage of focusing attention on the most recent actions making plan recognition in certain domains (e.g., web browsing) easier, but it has the side effect of reducing the complexity of reasoning over all the past actions to determine an agent's plan. Another way to handle this problem is to introduce uncertainty into the model. Jameson (1996) describes how the causal planning model can be used to construct Bayesian networks. The networks are constructed based on observable events within each phase of the model.

2.4 Decision and Utility Theory

The world we live in demands we make decisions about many things; from whether to get up in the morning to work, to what clothes to wear, to what to eat. Our decisions have consequences, some are more desirable than others. The preference of one consequence over another is based on personal preferences. Furthermore, certain consequences are more likely than others. Therefore, to make informed decisions, taking into account all preferences and factors affecting our decisions, we must assess the *utility* of all the outcomes of our decisions. Furthermore, we must know the chance of an outcome occurring given various background information we have.

The bodies of research known as Decision Theory and Utility Theory exists to assist decision makers, allowing them to make informed decisions. The Web Dictionary of Cybernetics and Systems[10] defines decision theory as follows:

> Decision theory is a body of knowledge and related analytical techniques of different degrees of formality designed to help a decision maker choose among a set of alternatives in light of their possible consequences ...In a decision situation under certainty the decision maker's preferences are simulated by a single-attribute or multi-attribute value function that introduces ordering on the set of consequences and thus also ranks the alternatives ...The decision maker's preferences for the mutually exclusive consequences of an alternative are described by a utility function that permits calculation of the expect utility for each alternative. The alternative with the highest expected utility is considered the most preferable ...[the] approach is to reduce the uncertainty case to the case of risk by using subjective probabilities, based on expert assessments or on analysis of previous decisions made in similar circumstances.

Given the above definition, utility theory's relationship to decision theory is to provide the semantics of and justification for determining the single-attribute or multi-attribute value function denoting a preference of one consequence of an alternative over another consequence. A decision maker's preference for a consequence is determined by many factors: the alternative being considered, the biases and/or prejudices of the designer, psychological and sociological factors, engineering considerations, etc. Any or all of these attributes may impact the final decision. The likelihood (i.e., probability) of a particular consequence does not affect a decision maker's desire (i.e., utility) for a consequence. However, the likelihood of a consequence combined with the consequence's desirability affects the decision maker's expected utility of that consequence. The relationship between preference and probability is succinctly stated by Pearl (1988, pg. 289).

> ...judgments about the likelihood of events is [sic] quantified by probabilities, judgments about the desirability of action consequences are quantified by utilities.

---

[10]See http://pespmc1.vub.ac.be/ASC/UTILITY.html

46

Rational decision makers adhere to the concept of the maximum-expected-utility (MEU) criterion. The MEU criterion states a decision maker will choose the alternative that maximizes his/her expected utility. This section proceeds to give some background on the MEU criterion. Unfortunately, there is no standardized notation in decision theory for mathematically stating the foundations of the maximum-expected-utility criterion. However, the notation used throughout this research is internally consistent, understandable, and generally accepted within the uncertainty in artificial intelligence community. Where appropriate, alternative notation is also given to be inclusive of other research communities' notation.

Let $\mathcal{A}$ be a non-empty, finite collection of alternatives. For each alternative, $\alpha \in \mathcal{A}$, there will be a finite collection of consequences $\alpha_{\mathcal{C}} \subset \mathcal{C}$, where $\mathcal{C}$ is the collection of consequences. Let $\mathbf{c} \in \alpha_{\mathcal{C}}$ denote a consequence of alternative α. Consequence $\mathbf{c}$ is an n-dimensional vector of attribute values

$$\mathbf{c} = (c_1, \ldots, c_i, \ldots, c_n). \tag{1}$$

For every attribute C_i, define the function $C_i : \mathcal{A} \mapsto \Re$ mapping the alternative α to some consequence space such that $C_i(\alpha) \equiv c_i$. Under uncertainty, the decision maker will not know which consequence will occur, but it is assumed he can determine the probability of the consequence occurring, given a probability distribution A, evidence $\mathbf{E}$, and any background information ξ. $\mathbf{E}$ includes the alternative, α, as well as any other observations. ξ includes *a priori* evidence such as previous decisions. Assuming A, $\mathbf{E}$, and ξ are given, let $\Pr(\mathbf{c}|\mathbf{E}, \xi)$ denote the probability of consequence $\mathbf{c}$ given evidence $\mathbf{E}$ and ξ[11]. Let $u(\cdot)$ be a positive real-valued utility function. Let $u(\mathbf{c})$ denote the value of the utility function of consequence $\mathbf{c}$. We can now define an

---

[11]Strictly speaking, $\mathbf{c}$, $\mathbf{E}$, and ξ are *random variables*.

expected utility function, $EU(\alpha)$[12], as follows:

$$EU(\alpha) \equiv \mathcal{E}_A[u(\mathbf{c})] = \sum_{\mathbf{c} \in \alpha_C} \Pr(\mathbf{c}|\mathbf{E}, \xi)u(\mathbf{c}), \tag{2}$$

where the expected utility function is equivalent to the expectation operator $\mathcal{E}_A$ taken with respect to probability distribution A. If Equation (2) is evaluated for all alternatives (also termed a *lottery*), the decision maker obtains a rank ordering of the expected utilities of the alternatives. Rationally, the decision maker chooses the alternative with the maximum (highest) expected utility.

Given the discussion above, every preference pattern of one lottery over another can be encoded by specifying the utility measure of each individual consequence and deciding all preferences among the lotteries on the basis of the MEU criterion. Alternatively, if a decision maker rationally chooses the lottery with the highest expected utility, he/she is guaranteed the choice is consistent with his/her utility for that lottery, regardless of the utility assigned to the consequences.

The utility function $U(\mathbf{c})$ can take into account psychological factors, such as the decision maker's biases, prejudices, preferences, mental state, etc. as well as aversion to risk. Note that Equation (2) is expressed as a summation due to the assumption that each consequence is mutually exclusive of the others for a given alternative. This assumption is realistic in a large number of real world problems (Keeney and Raiffa 1993). As a result, the following basic axiom from probability theory is used

$$\Pr(\mathbf{c_1} \vee \ldots \vee \mathbf{c_n}) = \Pr(\mathbf{c_1}) + \ldots + \Pr(\mathbf{c_n}), \tag{3}$$

where $\mathbf{c_1}, \ldots, \mathbf{c_n}$ are mutually exclusive, i.e., $\mathbf{c_1} \wedge \ldots \wedge \mathbf{c_n} = \emptyset$.

The decision theory analysis paradigm can be decomposed into the following five-step analysis process (Keeney and Raiffa 1993):

---

[12]The notation for the expected utility function is $EU(\cdot)$. $EU(\alpha)$ shall be used throughout this dissertation since the notation simplifies the presentation of the function.

1. **Pre-analysis.** A decision maker who is undecided about the course of action he/she should take in a particular problem identifies viable action alternatives.

2. **Structural Analysis.** The decision maker structures the problem by determining what choices can be made now, what choices can be deferred, what experiments can be performed, what information can be gathered, etc. These questions can be represented by a decision tree (Fig. 1). The decision tree contains nodes that are under the control of the decision maker termed *decision nodes* (i.e., the square nodes) and nodes that are not under the decision maker's full control termed *chance nodes* (i.e., the circled nodes).

3. **Uncertainty Analysis.** The decision maker assigns probabilities to all child branches of the chance nodes. These assignments can be made with any number of techniques, including expert judgment, empirical data, and personal preferences (biases) of the decision maker.

4. **Utility or Value Analysis.** The decision maker assigns utility values to the consequences associated with paths through the tree. Associated with each path are various economic and psychological costs and benefits that affect the decision maker and possibly others whom the decision maker considers part of the decision making problem. The utilities, therefore, represent a decision maker's preferences and are encoded as cardinal utility numbers. The utility numbers reflect both a ranking between different consequences and relative preferences for certain alternatives over the possible consequences.

5. **Optimization Analysis.** The decision maker calculates his optimal strategy — the one that maximizes the decision maker's expected utility. The strategy determines what choice the decision maker should make at every decision node.

2.4.1 Assessing Multi-Attribute Utility Functions. Decision theory posits that using the concept of maximum expected utility, a rational decision maker will choose the optimal strategy (i.e., alternative) to maximize the expected utility. Most

Alternatives   Consequences   Utilities

$$C'_1 \longrightarrow U'_1$$
$$C'_i \longrightarrow U'_i$$
$$C'_m \longrightarrow U'_m$$

A'

A''

$$C''_1 \longrightarrow U''_1$$
$$C''_j \longrightarrow U''_j$$
$$C''_n \longrightarrow U''_n$$
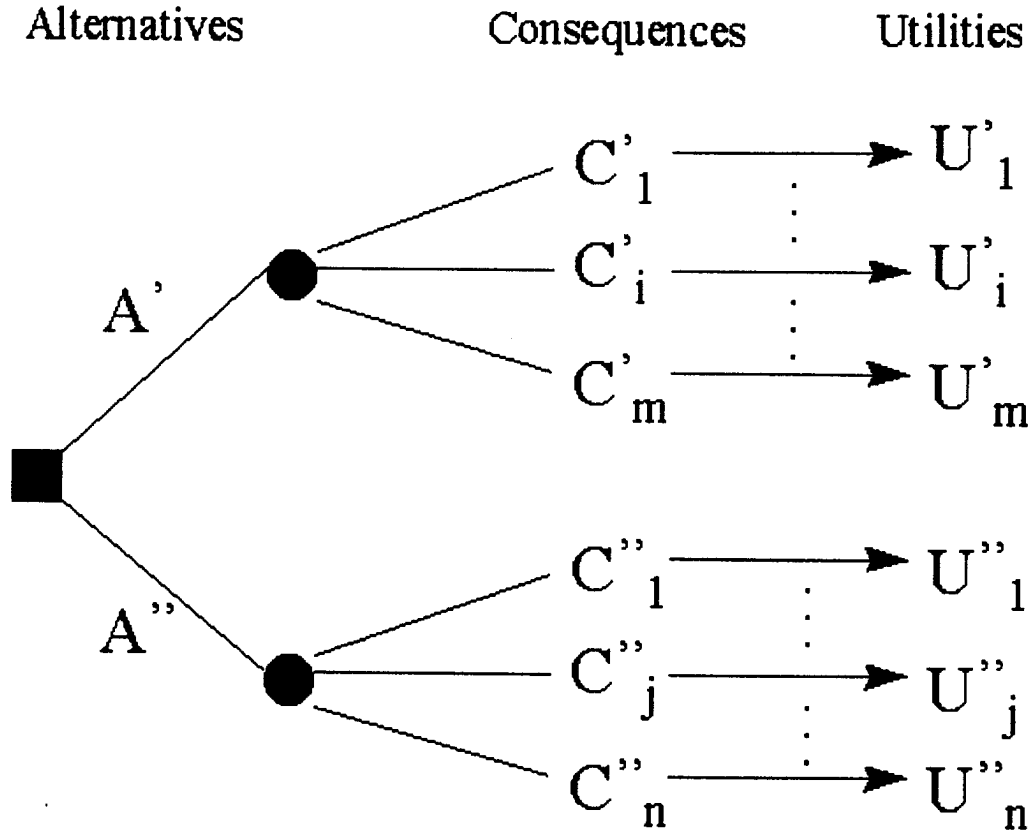
Figure 1   A Decision Tree Showing Two Alternatives.

decision makers do not know explicitly their utility function. Despite this fact, humans are capable of making decisions. Researchers have been concerned with the assessment of a decision maker's multi-attribute utility functions. However, the assessment of multi-attribute utility functions under uncertainty can be an arduous task.

The maximum expected utility criterion (also called the *principle of optimality*) allows one to compute the utility functions (also called the *optimal decision rule*) by backward induction of a decision tree representation of the decision space starting at a terminal leaf (i.e., consequence). Rust (1996) investigates empirically whether the main mathematical method for solving sequential decision problems under uncertainty, *dynamic programming* (Bellman 1957), provides a good model of

the way humans actually solve problems. Associated with dynamic programming is the concept of *structural estimation*. This "inverse stochastic control problem" utilizes empirical data of a decision maker's decision at some time t given the state of the world at time t to uncover the decision maker's utility function. The main problem with dynamic programming (and therefore structural estimation) is what Bellman (1957) calls "the curse of dimensionality." "The curse of dimensionality" simply states that the computational effort to compute the decision maker's utility function increases exponentially with regards to the number of states and decision variables as well as the number of time histories.

One way to reduce the amount of computation is to consider problems where the decision maker's next decision is dependent only on the last time history (Rust 1996). This myopic method is known as a (discrete-time) *Markov decision process*. Alternatively, some researchers take an expert systems approach to assessing the utility functions. For example, Horvitz and Barry (1995) elicit the utility functions from knowledge experts for decision-theoretic control of displays for a time-critical monitoring application at the NASA Mission Control Center. Horvitz and Rutledge (1991) studied the assessment and custom-tailoring of utility functions for time-dependent action. The authors state,

> Decision-theoretic methods have been considered inapplicable for general problem solving because they require agents to possess a utility function that provides a preference ordering over outcomes of action, and to have access to a probability distribution over outcomes associated with each decision.

They overcome this problem within their domain, a system constructed to explore the ideal control of inference by reasoners with limited abilities, by using utility functions assessed by experts for prototypical situations and *models* of time-dependent utility. The prototypical utility functions are modified by applying a mathematical model. Some preliminary results for determining utility functions from users exist (Druzdzel and van der Gaag 1995; Ha and Haddawy 1997; Shoham 1997).

2.5 Bayesian Networks

Bayesian networks are a probabilistic knowledge representation used to represent uncertain information (Pearl 1988). The directed acyclic graph structure of the network contains representations of both the conditional dependencies and independencies between elements of the problem domain. The knowledge is represented by nodes called random variables (RVs) and arcs representing the (causal) relationships between variables. The strengths of the relationships are described using parameters encoded in conditional probability tables.

One of the criticisms of Bayesian networks is how they are constructed. Construction of Bayesian networks usually proceeds in one of two ways: "hand-coded" or "machine-coded." "Hand-coded" means a Bayesian network is constructed by a knowledge engineer to capture the relevant random variables and their relations in the domain. The structure and parameters are encoded into a (typically) static Bayesian network. As with most other knowledge based systems, the knowledge acquisition process is the "bottleneck" of building a system around Bayesian networks. "Machine-coded" Bayesian networks, on the other hand, are generated by using previously captured data or data acquired as users interact with a system and therefore attempt to avoid the knowledge acquisition bottleneck. This data can either be "raw" data or another knowledge representation that is then transformed into a Bayesian network. Of course, this method of construction assumes access to the data. In many instances raw data is not available. For data captured in another knowledge representation, the knowledge acquisition bottleneck problem merely shifts to the new representation.

Whether knowledge engineers construct a Bayesian network *a priori*, using machine learning techniques, or a combination of the two, they must deal with the issue of causality. Pearl contends humans are good at specifying intuitive causal relationships between random variables (Pearl 1988). Jameson (1996, pp. 200–201) considers the direction of the causal relationships between general and specific abil-

ities. He notes that when a Bayesian network is constructed from general to more specific abilities, knowledge (i.e., evidence) about one specific ability typically propagates upward and then downward, increasing knowledge of more general abilities as well as other specific abilities. On the other hand, networks constructed from specific to general abilities typically only propagate knowledge downward, yielding no new knowledge about other specific abilities. Several other researchers have investigated issues concerning the construction of Bayesian networks (e.g., (Henrion 1989; Druzdzel and van der Gaag 1995; Pradhan et al. 1995)).

While the Bayesian network representation can concisely represent large amounts of knowledge, it also creates difficulties in manipulating the networks. In spite of the independence explicitly encoded into the network, many inferencing methods must search through the entire combinatoric space of all possible instantiations to the nodes of the network. It has been shown that this is problem is non-polynomial NP-hard (Cooper 1990).

Many methodologies for making Bayesian networks computationally feasible exist. The first is to *approximate the inferencing scheme* used to calculate updated beliefs in the network. Approximation methods for belief updating include stochastic search (Cousins 1991), junction trees (Lauritzen and Spiegelhalter 1988; Kjaerulff 1995), simple Genetic Algorithms (GAs) (Michalewicz 1992) and incremental or asymptotic evaluation methods such as Local Partial Evaluation (Draper and Hanks 1994). The second method involves *approximating the structure* of the Bayesian network itself. Methods for approximating the structure include removal of weak dependencies (Kjaerulff 1994) and removal of arcs (Sarkar and Murthy 1996). Another method uses only the relevant portions of the network with relationship to the evidence and query nodes (Breese and Heckerman 1996). Alternatively, goals, actions, pre- and post-conditions can be represented in a probabilistic knowledge base. Techniques for constructing Bayesian networks from probabilistic knowledge bases abound (Bacchus 1993; Goldman and Charniak 1993; Haddawy 1994).

Druzdzel (1997) discusses five useful properties of probabilistic knowledge representations making them desirable to use in our situation. In particular,

1. Directed probabilistic graphs capture essential qualitative properties of a domain, along with its causal structure.

2. Concepts such as relevance and conflicting evidence have a natural, formally sound meaning in probabilistic models.

3. Probabilistic schemes support sound reasoning at a variety of levels ranging from purely quantitative to purely qualitative levels.

4. The role of probability theory in reasoning under uncertainty can be compared to the role of first-order logic in reasoning under certainty.

5. Probabilistic knowledge representations support automatic generation of understandable explanations of inference for the sake of user interfaces to intelligent systems.

2.6 Summary

This research builds on the foundations laid by three separate research fields — human-computer interaction, artificial intelligence, and user modeling — concerning interface agent research. The orthogonal approach of each field presents a number of strengths. Furthermore, the relevant weaknesses of the three research fields have been given. These weaknesses have served to provide areas for additional research. While researchers in each field are cognizant of the strengths and weaknesses in their own field, this background has served to introduce researchers in any one field the strengths and weaknesses of the other two. This chapter has also served to summarize relevant background on decision theory and Bayesian networks.

III. Philosophy of Offering Assistance

This chapter presents the foundational philosophy for an approach offering timely, beneficial assistance. Specifically, the underlying philosophy for ascribing user intent and how this philosophy can be used by an interface agent acting as a decision maker for a user is discussed. Furthermore, this chapter discusses how interface agents play a key role in symbiotic information retrieval and decision support, a new approach for collaborative, decision-theoretic assistance.

3.1   User Intent Ascription — From Goals to Actions

Recall that the definition this dissertation uses for *user intent* is "the actions a user intends to perform in pursuit of his/her goal." To ascribe user intent, interface agent designers must identify the salient characteristics of a domain environment and specifically determine goals a user is trying to achieve, the reason and/or cause for pursuing those goals, and the actions to achieve those goals (Brown, Santos Jr., Banks, and Oxley 1998). This approach is based on the belief that what a user intends to do in an environment is the result of environmental events and/or stimuli occurring in the environment and by the goals they are trying to obtain as a reaction to the events and stimuli. That is, the reason *why* users perform actions is to achieve goals they pursue as a result of environmental stimuli. Social scientists support this belief about why we act. To quote from Pestello and Pestello (1991):

> we act, either verbally or overtly, in response to the symbolic meaning the confronting object has for us in the given situation.

Social scientists use intentions (as determined by surveying subjects) to measure possible future behaviors (i.e., actions) (Pestello and Pestello 1991). That is, what a user says he/she intends to do is indicative of what he/she really might do. They note that intentions do not necessarily translate into action, i.e., ascribing user intent is an uncertain process. The philosophy used in the research in this dissertation, on the

other hand, is to observe behavior (and other environmental events) in an attempt to ascribe a user's intent so as to predict future behavior.

Benyon and Murray (1993) use the term *task* or *intentional level* to describe the component of a user model containing knowledge about the user's goals. The task level knowledge is used to infer the goals the user is pursuing. They state that "failure to recognize the intentions underlying some user action will result in less satisfactory interaction" as this results in failing to recognize the pursuit of one goal versus another.

To achieve a goal a user must perform certain actions. Goals can be composed of multiple actions with many pre- and post-conditions. Pre-conditions are directly observable events in the environment (e.g., nodes A and B are not mutually exclusive, sensor A7 is non-operational, the plane's altitude is 10,000 ft. above sea-level). These pre-conditions cause a user to pursue a goal and/or affect the goal a user will pursue. Additionally, other factors affect the goals a user pursues as well as the actions the user will take to achieve the goal. In particular, human factors (e.g., skill, work load, expertise, etc.) all affect the user's decision to pursue goals and perform actions in pursuit of goals. Typically, these factors are not directly observable but they are measurable, either *a priori*, such as skill or expertise, or dynamically as the user interacts with the environment, e.g., work load.

A directed acyclic AND/OR graph shows causality between the pre-conditions, goals, and actions. For AND goals, all the actions must be performed to achieve the goal. For OR goals, only one action is needed to achieve the goal. Similarly, pre-conditions for a goal may all have to be present (AND), or one or more may need to be present (OR). Other possible relationships can exist. For example, an XOR relationship would represent the case where only one pre-condition or action can be "active" for a goal. For example, Figure 2 shows an OR goal and several sub-goals. Pre-conditions in this figure are the roots of the tree and the actions are the leaves of the tree.
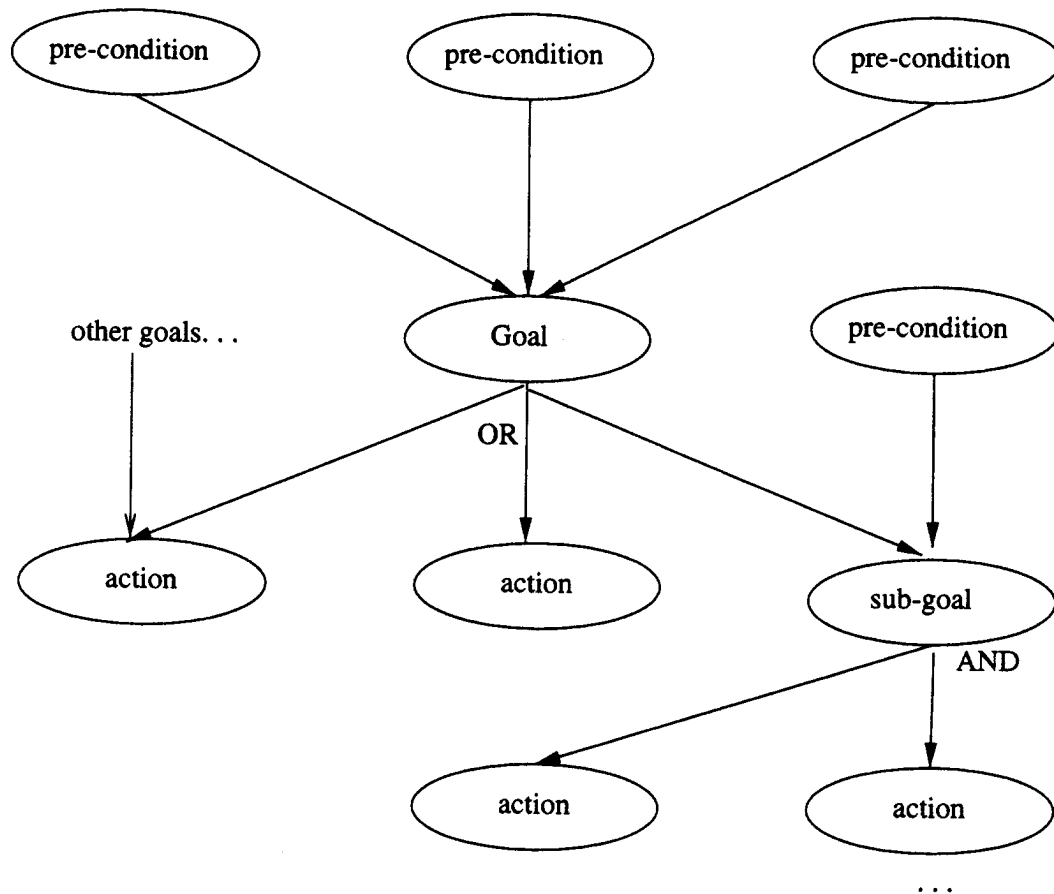
Figure 2   A directed acyclic graph representation of a user model.

There are several advantages to representing users' intentions in a goal hierarchy, as represented by a directed acyclic AND/OR graph, such as the following:

- Goal abstraction allows one to design and detect higher level goals, in pursuit of lower level goals.

- Keyhole plan recognition (see Section 2.3) is made easier by explicitly enumerating pre- and post-conditions and atomic actions composing goals (Albrecht et al. 1997; Waern 1996).

- Natural language explanations of actions based on prediction of goals can be easily generated from the structure.

3.2 Approach — Symbiotic Information Reasoning and Decision Support

The dissertation hypothesis is part of an encompassing hypothesis that serves as a foundational framework for the entire dissertation. This framework hypothesis is stated as follows:

> A holistic, decision-theoretic methodology to the interface agent development life cycle addresses the need for symbiotic information reasoning and decision support.

The methodology takes a human-centered approach to task partitioning, where the computer (i.e., interface agent) is responsible for abstracting and analyzing information from the user's environment to enable the user to understand the relevant information and performing necessary analysis to allow the system to provide information highlighting and user focus of attention activities. The methodology addresses the entire development life cycle of an interface agent (and its user model), from requirements to maintenance.

The principle of Symbiotic Information Retrieval and Decision Support (SIRDS) as an approach for collaborative, decision-theoretic assistance was first presented at the 19th Interservice/Industry Training Systems and Education Conference (Banks, Stytz, Santos Jr., and Brown 1997). Focusing on strengths of the artificial intelligence, human-computer interaction, and user modeling research communities, the goal of the SIRDS approach is to develop comprehensive software engineering, knowledge engineering and acquisition methodologies, principles, heuristics, and tools for symbiotic information reasoning and decision support.

The name for the approach is descriptive of its intent. The interface should operate symbiotically; that is, work tasks should be appropriately partitioned between the computer and the user. The computer's strength lies in its ability to perform data acquisition and management (to include display of this information (Horvitz and Barry 1995)) from many heterogeneous sources, low level quantitative and qualitative data analysis, and routine inference to enable decision support. A user's

strength lies in the ability to provide guidance and insight concerning the information that is necessary to draw complex, higher level inferences from the data. A symbiotic approach is necessary because the objective is to let the user and the computer share the task load; therefore, a human-centered approach to task partitioning is used.

The information reasoning component of the approach deals with issues related primarily to abstracting and analyzing information. The decision support aspect relates to the need to enable the user to understand the relevant data and to perform necessary analysis to allow the system to provide information highlighting and user focus of attention activities.

In addition to performing information retrieval and analysis, SIRDS uses information visualization techniques to enable the user to understand the derived information, synthesis operations, and available processing options. However, to maximize user effectiveness in an information dense environment, the approach must operate in anticipation of user information needs. To do so, we must first ascertain user information requirements based on the current situation and a history of required information. The SIRDS approach then initiates data retrieval operations and provides focus of attention on and analysis of the resulting relevant information.

SIRDS requires the development of an adaptive, intelligent, learning human-computer interface. Construction of the interface requires a mix of traditional human-computer interaction techniques, data visualization, and intelligent agents. To be sure, intelligent agents are a key aspect of the SIRDS approach. They perform information fusion, analysis and abstraction, as well as deriving information requirements and controlling information display.

An interface agent, possessing knowledge of the environmental stimuli, user goals, actions, and various human factors can act as an assistant, offering assistance on behalf of the user. This knowledge can be captured in a user model. Specifically, a knowledge representation capable of capturing the causal relationship between the

stimuli, goals, and actions must be used. Furthermore, since ascribing user intent is inherently uncertain, the knowledge representation should be capable of representing the uncertainty in a sound, non-*ad hoc* manner. Bayesian networks meet both of these criteria. The next chapter discusses how an interface agent can use knowledge of a user's goals and the current environmental state to act as a decision maker on behalf of the user.

3.3 Conclusion

The philosophy and the symbiotic information reasoning and decision support approach presented in this section are foundational to the construction of a dynamic, uncertainty-based knowledge representation for user intent ascription. Previous research in the fields of user modeling (Benyon and Murray 1993; Jameson 1996; Waern 1996) and social science (Pestello and Pestello 1991) lend support to this foundation. Experiments performed also corroborate this foundation (see Chapter VII). Interface agents, a key aspect of the SIRDS approach, using this foundation are able to provide assistance to a user based on the goals the user is pursuing, taking into account human factors that affect the user's decision to pursue goals and perform actions in pursuit of goals.

IV. Decision Theory-Based Approach

The previous chapter introduced the philosophy underlying the relationships between pre-conditions, goals, and actions as well as SIRDS as an approach for collaborative, decision-theoretic assistance. This chapter presents the Core Interface Agent (CIA) architecture. The CIA architecture implements the philosophy for user intent ascription and is based on the symbiotic information retrieval and decision support approach and functions as an autonomous decision maker, acting on behalf of the user.

The details of the entire architecture are developed over several chapters. Specifically, this chapter focuses on the decision making capabilities of the interface agent. The chapter describes how a Bayesian network-based user model (capturing the probabilistic causality between pre-conditions, goals, and actions) and associated utility functions (capturing a user's utility for having the interface agent perform an action on the user's behalf to achieve a goal) can be used in a decision-theoretic manner to offer assistance to a user. Chapter V presents the requirements levied on the architecture and the associated metrics to measure the agent's ability to meet those requirements. Furthermore, Chapter V discusses how these metrics can be combined into a requirements utility function that can be used to determine when the agent is not meeting the requirements. Finally, Chapter V describes how a multi-agent system of correction adaptation agents can be used to correct an inaccurate user model as determined by the requirements utility function. Chapter VI discusses how the CIA architecture fits into an agent development environment. The elicitation of the user model and utility functions is best discussed in Chapter VII, which discusses in detail the three systems used for experimentation.

4.1 Core Interface Agent Architecture

The Core Interface Agent (CIA) architecture is a multi-agent system composed
of an interface agent and a collection of correction adaptation agents. The purpose
of the architecture is to provide assistance to the user. This purpose is accomplished
by maintaining an accurate model of the user's interaction with the target system
environment. The user model is used to ascribe the user's intent. Figure 3 gives an
overview of the CIA architecture. The task of ascribing user intent is delegated to
the interface agent component of the architecture, while continual adaptation of the
interface agent's user model is a task shared by the interface agent and the collection
of correction adaptation agents.
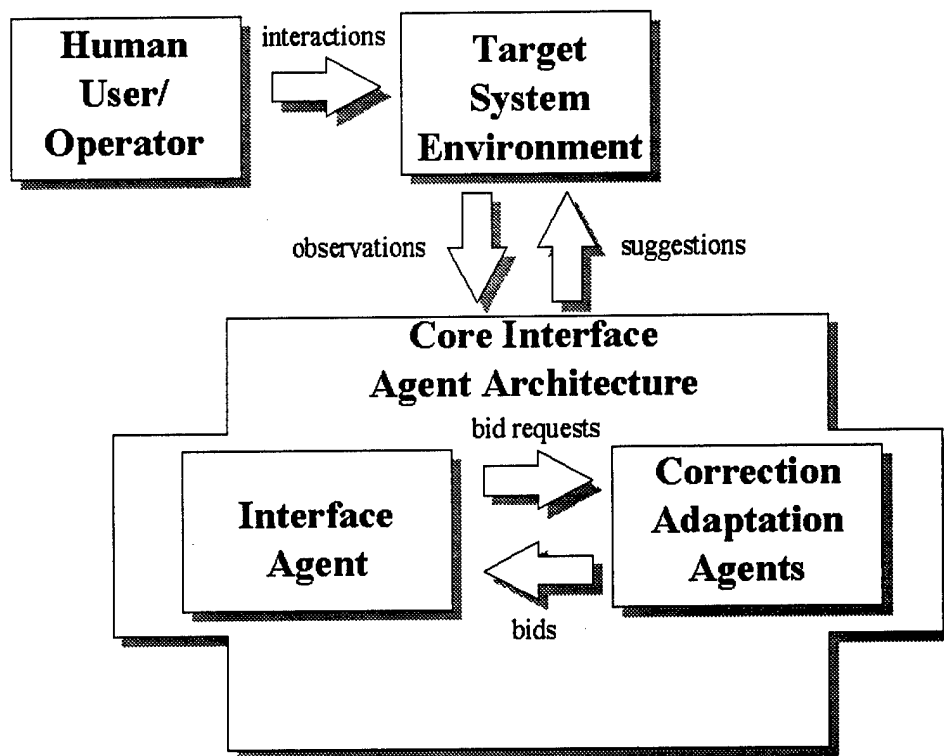
Figure 3   Core Interface Agent Architecture: Process flow diagram for user intent
prediction and continual adaptation of an intelligent user interface agent.

A user interacts with a target system, typically a direct manipulation interface[1].
This interaction — the menus chosen, the buttons pressed, the text typed — as well
as other target system environment stimuli (e.g., a spelling error, the arrival of a
new e-mail message) are communicated to the CIA architecture as observations.
These observations are used by the interface agent to infer what a user is doing
within the environment and to ascribe the user's intent. A keyhole plan recognition
approach is used, where the human is unaware of or indifferent to the user intent
ascription process. Based on the knowledge of the environment that the interface
agent possesses and the user's interaction with the environment, the interface agent
determines the goal with the highest expected utility and offers a suggestion to
the user via the target system. If the interface agent determines its user model is
inaccurate, it begins a bidding process with the correction adaptation agents. The
correction adaptation agents offer "bids" to modify the interface agent's user model.
The interface agent allows the correction adaptation agent offering the best bid to
modify the user model.

Figure 4 shows the architecture of the interface agent and the correction adap-
tation agents. The architectures for the interface agent and correction adaptation
agent are the same. The difference between the two is the implementation of the cor-
rection model, discussed in Chapter V. The agents, as well as the target system, all
communicate via the Knowledge Query and Manipulation Language (KQML) (May-
field, Labrou, and Finin 1996). KQML is both a message format and message-
handling protocol, supporting run-time sharing of knowledge between heterogeneous
(and homogeneous) agents. KQML is emerging as a standard within the agent com-
munity and a number of tools exist (to include application programming interfaces,
or APIs) to support KQML integration into an agent-based application.

Each target system observation (environmental stimuli or user action) is com-
municated to the agents via the KQML message passing API. Every observation is

---
[1]See Chapter VII for a system using a natural language user interface.

63
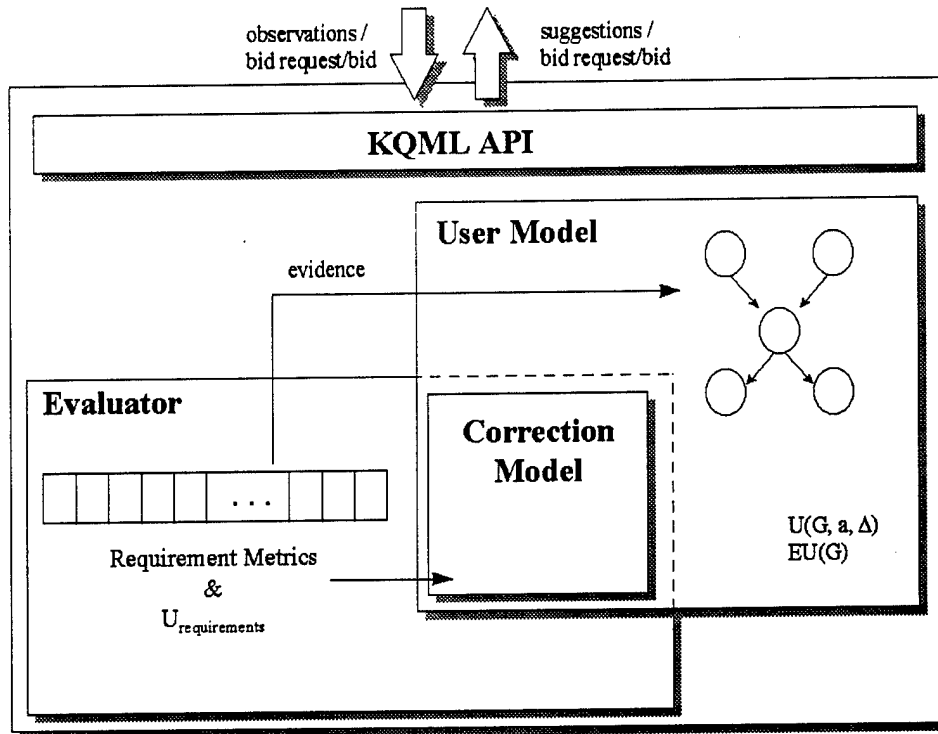
Figure 4   Interface Agent and Correction Adaptation Agent Architecture.

stored by the agents' evaluator in a history stack (i.e., most recent observation is on the top of the stack). These observations are used by the agents as evidence in the Bayesian network-based user model. Evidence may "fade" over time, essentially allowing the interface agent to "forget" past observations. The architecture supports (possibly) unique fading functions for each observation. The types of fading functions supported include a time-based fading function (evidence is relevant for a specified time) and a queue-based fading function (only the N newest observation are relevant).

The user model is composed of three components: the Bayesian network user model, a utility model, and a user profile. The Bayesian network user model captures the uncertain, causal relationship between the pre-conditions, goals, and actions. The utility model contains the utility functions for the attributes (i.e., human factors) and the additive multi-attribute utility function combining the other utility functions. The utility functions capture a user's utility for having the interface agent

perform an action on the user's behalf to achieve a goal. The user profile captures knowledge about the user including background, interests, and general knowledge about the user that is typically static. Two user defined thresholds, one for offering (collaborative) assistance and one for autonomously performing actions on the user's behalf to obtain a user's goal, determine how/if the interface agent will offer assistance. This approach is the same as the one presented by Maes (1994a), except she based her thresholds on statistical probabilities and the ones in this research are based on the expected utility function. The user profile also contains the values of any *static* human factors (e.g., skill, spatial memory). The normalized values of these human factors can be determined off-line and do not change as the user interacts with the target system.

4.1.1 Offering Assistance. The interface agent determines what type of assistance to offer a user by calculating the expected utility of offering assistance for a goal, $EU(\alpha_G)$. This calculation occurs both periodically and whenever new evidence is observed[2]. The target system is responsible for displaying the suggestions to the user and/or performing the suggested actions. A suggestion is offered only if the expected utility exceeds a user chosen threshold for offering assistance. As implemented, the interface agent autonomously performs the actions associated with the highest ranked goal (based on its expected utility) if the goal's expected utility is greater than the autonomy threshold[3]. Otherwise, if the goal's expected utility is above the collaborative assistance threshold, the interface agent sends a request (i.e., a KQML message) to the user to perform on the user's behalf the actions that would achieve the goal. Otherwise, the interface agent offers no assistance.

---

[2]The periodic calculation allows the interface agent to offer assistance even if the environment is static, e.g., the user is not performing actions. This may be helpful if, for instance, the user does not know which actions to perform and performs no actions as a result.

[3]In actuality, the interface agent generates and sends a KQML message to the target system to perform all the non-observed actions associated with the highest ranked non-observed goal. The "non-observed criteria" prevents the interface agent from suggesting non-faded (i.e., recently observed) goals and associated actions.

There are a number of other ways to decide the goals for which to offer assistance. Among the possible alternatives, the interface agent can suggest any goal above the collaborative threshold *and/or* perform those actions associated with all goals above the autonomous threshold value. A slight variation can limit the number of suggestions made and actions performed. This would be useful for target systems with many goals whose expected utility values were significantly close. However, the implemented method for suggesting assistance is better than the alternatives for the following reason: any assistance actually performed is added to the evaluator's history stack as an observation. These new observations affect the updated belief probabilities and therefore the expected utility ranking of the other goals. Therefore, a myopic assistance suggestion method is better. However, future versions of the CIA architecture may allow the agent to collaboratively suggest more than the top ranked goal.

There are three underlying methodologies utilized in the CIA architecture. These methodologies are a decision theory-based approach for assistance (presented next), requirements metrics (Section 5.1), and the correction model and associated correction adaptation agents (Section 5.3).

4.2   Decision-Theoretic Assistance

Utility theory, using Bayesian techniques for assessing the probabilities, is a non-*ad hoc* approach for predicting user intent. The user's utility function can take into account *relevancy* of the goal with respect to any number of attributes and/or *discriminators* in the environment. These attributes tell us what is important, explicitly enumerating those factors that impact the utility of choosing the goal. The attributes may take into account the psychological factors, such as the user's cognitive load or preferences; system factors, such as processor load, explicit requirements placed on the system by the designer (e.g., reaction time); or simply the goal at hand. For the attributes used, see Appendix B.

66

An interface agent can use the utility function to determine the following:

- **What action to take** — the ones with the highest expected utility;

- **When to take action** — when the expected utility is above some user-defined threshold;

- **Why to take an action** — the action helps the user achieve the goal they are pursuing; and

- **How to take an action** — the action itself.

The user model component of the CIA architecture (see Figure 4) is responsible for storing the user's utility function $U(G, a, \Delta)$ as well as performing the expected utility calculations for determining the type of assistance to offer the user. $EU(\alpha_G)$ is calculated by using the Bayesian network user model to perform belief updating on the goal random variable and the utility of suggesting the actions used to achieve the goal. The remainder of this section presents the details of the CIA architecture's approach to offering decision-theoretic assistance.

Let $\mathcal{A} = \{autonomous, collaborative, none\}$ denote the set of types of assistance the interface agent can offer the user for $\mathcal{G}$, a finite collection of goals. For each goal $G \in \mathcal{G}$ there is a finite collection of actions $A_G \in \mathcal{A}_g$ for achieving that goal, where $\mathcal{A}_g \equiv \bigcup_{G \in \mathcal{G}} A_G$ is a finite collection of all possible actions the user can perform. Let $\alpha_G \equiv (\alpha, G) \in \mathcal{A} \times \mathcal{G}$ denote the type of assistance offered for goal G. Let Δ denote an n-dimensional vector of applicable discriminators that may also impact the utility (e.g., work load). In decision theory, these discriminators are termed attributes. This dissertation uses both terms. Δ denotes both the discriminators themselves and the vector of discriminator values. **E** denotes any observed evidence and ξ any background information on the user. The background information can be stored as a user profile, allowing persistent storage of various information inferred about the user. As in Equation (2), **E** and ξ are *random variables*, as are G and A_G. Every one of these random variables is a random variable (i.e., node)

in the Bayesian network user model. The Bayesian network also serves to define the probability distribution via a joint probability density function over the random variables.

Let $u(\cdot)$ be a positive real-valued multi-attribute utility function and $u(\alpha_G, \Delta)$ be the value of $u(\cdot)$, denoting the utility of offering assistance α_G for goal G, given the consequence Δ of this assistance. Specifically, $u(\alpha_{G=True}, \Delta)$ is the utility of offering assistance for the goal if the user is pursuing the goal (i.e., the assistance is right). $u(\alpha_{G=False}, \Delta)$ is the utility of offering assistance for the goal if the user is not pursuing the goal (i.e., the assistance is wrong). Let $\Pr(G = True|\mathbf{E}, \xi)$ denote the probability the user is pursuing goal G given evidence E and ξ. Conversely, let $\Pr(G = False|\mathbf{E}, \xi)$ denote the probability the user is not pursuing goal G given evidence E and ξ. Let the expected utility function, $EU(\alpha_G)$, be defined[4] as

$$
\begin{aligned}
EU(\alpha_G) &\equiv \mathcal{E}[u(\alpha_G, \Delta)] \\
&= \Pr(G = True|\mathbf{E}, \xi)u(\alpha_{G=True}, \Delta) \\
&\quad + \Pr(G = False|\mathbf{E}, \xi)u(\alpha_{G=False}, \Delta).
\end{aligned} \tag{4}
$$

Since a goal is achieved by performing actions, the utility function $u(\alpha_G, \Delta)$ can be further decomposed. There are two decompositions we must consider since the goal-action graph is represented with an AND/OR graph. For AND goals, all the actions must be performed to achieve the goal. Given an action $a \in A_G$, let $U(\cdot)$ be a positive real-valued utility function and $U(G, a, \Delta)$ the value of the function denoting the utility of action a with respect to goal G and the value of the discriminators, Δ. Let $\Pr(a|\mathbf{E}, \xi)$ denote the probability of action a given evidence E and ξ. We can now further refine $u(\alpha_G, \Delta)$ as follows:

$$
u(\alpha_G, \Delta) = \sum_{a \in A_G} \Pr(a|\mathbf{E}, \xi)U(G, a, \Delta). \tag{5}
$$

---

[4] $\mathbf{E}$ and ξ are suppressed from the equations.

68

For OR goals, the interface agent chooses the best action to achieve the goal. We define $u(\alpha_G, \Delta)$ as follows:

$$u(\alpha_G, \Delta) = \max_{a \in A_G} \Pr(a|\mathbf{E}, \xi) U(G, a, \Delta). \tag{6}$$

The interface agent acts as a rational decision maker, using the maximum-expected-utility criterion presented in Section 2.4. Equation (4) allows the interface agent to obtain a rank ordering over all possible assistance. This ranking is stored in the evaluator's history stack and can be used by the interface agent and correction adaptation agents (see Chapter V). As mentioned previously, the interface agent suggests the goal with the maximum (highest) expected utility taking into account the user-defined assistance thresholds. Figure 5 shows a decision tree for the interface agent for one goal. For the decision node in Figure 5, Equation (4) is evaluated for each of the types of assistance, $\mathcal{A}$.

4.3   User Model Construction

Construction of the Bayesian network user model and associated utility functions raises a difficult question: "how does a designer best construct the user model?" For environments where the user's goals and actions are relatively static, a designer can use any number of well-known knowledge elicitation techniques (Cooke 1994) along with methods for Bayesian network construction (discussed in Section 2.5). An approach for determining the goals, actions, and pre-conditions within an environment for adaptive systems is offered by Benyon and Murray (1993). They point out five analysis phases that must be considered when designing adaptive systems. The first two, functional and data analysis, are analogous to the software engineering techniques of function-oriented and state-oriented problem analysis (Davis 1993). The third phase, task knowledge analysis, focuses on cognitive characteristics required of users by the system (e.g., cognitive loading). The next phase, user

Alternatives	Consequences	Utilities

right $\longrightarrow$ U'_{right}

autonomous

wrong $\longrightarrow$ U'_{wrong}

right $\longrightarrow$ U''_{right}

collaborative

wrong $\longrightarrow$ U''_{wrong}

no action

right $\longrightarrow$ U'''_{right}

wrong $\longrightarrow$ U'''_{wrong}
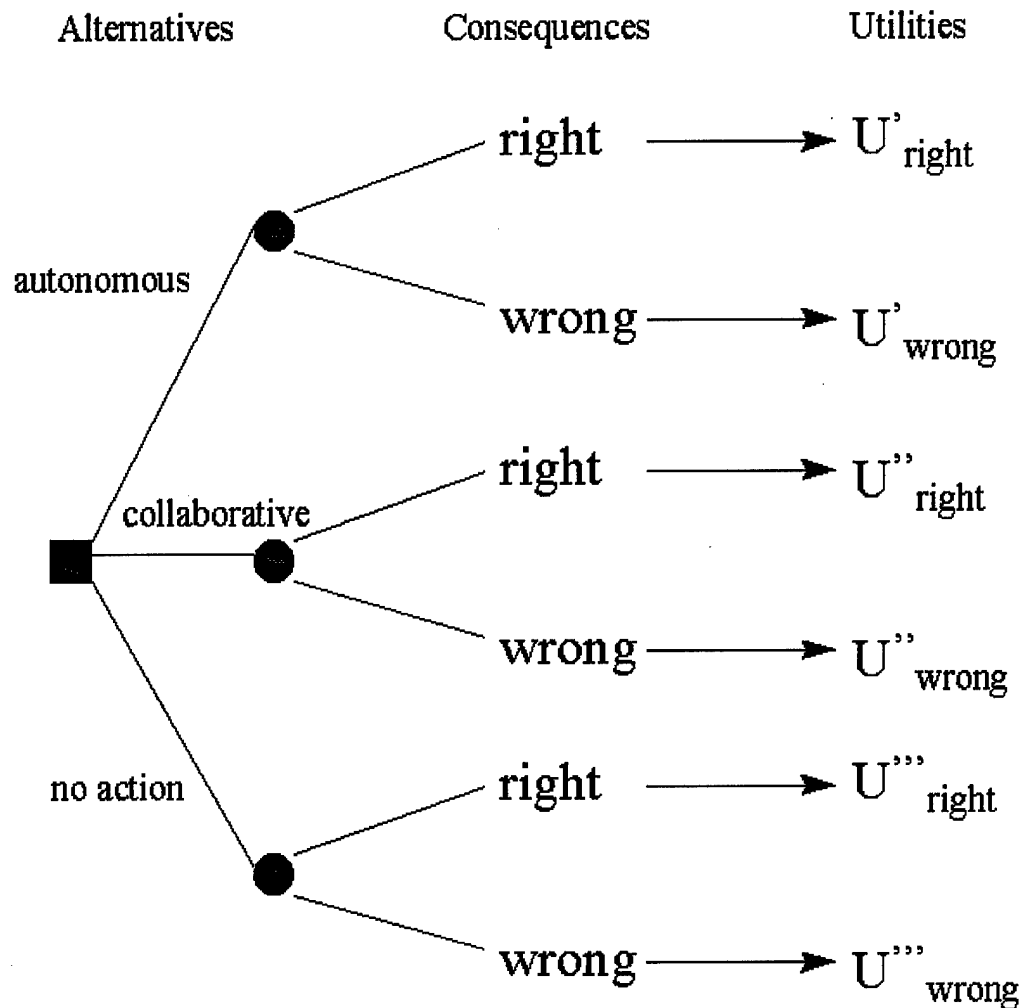
Figure 5   The Interface Agent's Decision Tree for One Goal.

analysis, determines the scope of the user population where the system is able to respond. This analysis is concerned with obtaining attributes of users of the application. The last phase, environmental analysis, is, obviously, concerned with the environment within which the system is to be situated in, including physical aspects of the system. This declarative approach was used for two of the three experimental domains (see Chapter VII).

Certain environments do not allow designers to fully specify the user model *a priori*. In these cases, the user model must be dynamic. Designers can therefore

consider research on learning Bayesian networks (Geiger and Heckerman 1995; Heckerman 1995; Heckerman and Geiger 1995; Sarkar and Murthy 1996; Friedman and Goldszmidt 1996; Chickering, Heckerman, and Meek 1997; Chu and Xiang 1997; Greiner, Grove, and Schuurmans 1997; Meek and Heckerman 1997; Ramoni and Sebastiani 1997) *and* utility functions (Rust 1996). Research on learning Bayesian networks has shown it is possible to learn both the structure and the parameters (i.e., conditional probabilities) of the network. Rust (1996) presents an overview of several researchers' work on computing users' utility functions by backward induction. Furthermore, Rust discusses the applicability of these learned utility functions as models of real users, stating that under certain circumstances, these utility functions serve as good approximations to a user's "real" utility function.

For the research presented in this dissertation, learning Bayesian network techniques have limited applicability. As a result of the user intent ascription philosophy presented in Chapter III, the structure of the learned Bayesian network is restricted to the pre-conditions to goals to actions philosophy. This forces pre-condition random variables in the Bayesian network to be parents of goal random variables. Likewise, goal random variables must be parents of action random variables. Existing techniques for Bayesian network structure learning do not account for this structure of the Bayesian network user model. Furthermore, these techniques do not account for the semantics of the AND/OR graph used in this research. These limitations do not prevent the learning of the conditional probabilities. The technique of learning of the conditional probabilities is used in this research to correct the user model.

Two factors allow us to bypass the limitations with existing structural estimation techniques for Bayesian networks: domain dependent information and the rank ordering over all possible assistance. Domain dependent information allows designers to dynamically build user models given known information about the types of interactions users have with the target system. Research performed by Horvitz et al. (1998) revealed classes of evidential distinctions, providing information about the

structure of the Bayesian network needed to make inferences about a user's problems and the user's need for assistance. For example, actions performed in succession over a short time period may be indicative of actions that may be associated with the same goal. See Section 2.3 for the classes Horvitz et al. identified. Several user model construction heuristics were identified during performance of this research and are discussed in Section 5.5. Using domain dependent information in this way is different than the declarative approach previously discussed. In the latter, interface agent user model designers use their domain expertise to *a priori* design the user model. In the former case, domain dependent information is used to procedurally "guide" the construction of the user model, based on a particular user's interactions.

The CIA architecture's rank ordering over all possible assistance makes it possible to modify the existing user model. For example, new goals can be added to the user model associating actions with the new goal that have high expected utility. Alternatively, actions with low expected utility for achieving a goal can be removed from that goal's set of achievable actions. The approach of using domain independent information and the CIA architecture's rank ordering over all assistance was used in one of the three experimental domains (see Chapter VII).

The problem with structural estimation of the utility functions is what Bellman (1957) calls "the curse of dimensionality." "The curse of dimensionality" states that the computational effort to compute the decision maker's utility function increases exponentially with regards to the number of states and decision variables as well as the number of time histories. Based on research on multi-attribute utility functions presented in Section 2.4, an expert systems approach was taken in this dissertation similar to the one presented by Horvitz and Rutledge (1991). The authors used utility functions assessed by experts for prototypical situations and *models* of time-dependent utility. The prototypical utility functions are modified by applying a mathematical model.

V. Interface Agent User Model Correction

As discussed previously, an accurate user model is considered necessary for effective prediction of user intent. However, there are many possible causes for an inaccurate user model. The plethora of machine-learning techniques employed in current user models attests to two facts: users are different and user behavior changes over time. Both of these facts are causes of the same effect: a deviation of user behavior from the originally specified and designed user model (assuming initially an accurate user model is captured). User models that fail to dynamically adapt to different users and the changing behaviors and/or needs of a user are doomed to be inaccurate in short order.

This chapter is organized as follows. Section 5.1 introduces four requirements for an interface agent. Associated with these requirements are a set of metrics that measure an interface agent's ability to meet the requirements. To account for the uncertainty and dynamics involved in predicting user intent and modeling user behavior, the set of requirement metrics can be combined into a requirements utility function that can be used to determine when and how to correct the interface agent's user model. Section 5.2 discusses several types of specific problems that may arise as the user's needs change over time. Section 5.3 discusses how a multi-agent system of correction adaptation agents can be used to correct an inaccurate user model using the requirements utility function. Finally, Section 5.5 describes a theory for correction adaptation agent evolution, showing how a collection of "atomic" correction adaptations can evolve into fully functional agents. This theory is useful in answering the question "which correction adaptation agents do we build?"

5.1 Interface Agent Requirements and Metrics

For the agent to perform within its environment, the agent must determine *what is important* to model in the domain, with associated discriminators and/or

73

metrics to determine and define *why*, *when*, and *how* to dynamically change the user model. To improve an interface agent's utility for providing assistance to the user, agent development must explicitly take into account the agent's requirements. Methods for determining when those requirements are not being met and how to correct the agent's user model is the primary focus of this section.

Many of the requirements for agents are not well defined and many of these requirements are not mutually exclusive. Furthermore, these requirements do not set agents apart from other forms of software (see Petrie (1996)). This shortfall apparently causes most researchers to ignore requirements for agent development.

This section explicitly considers requirements for interface agents. The section develops a well-defined, measurable set of requirements, with associated metrics, to determine if the interface agent is meeting the requirements. Each metric is categorized under the top-level requirement associated with it; however, a clear delineation between the requirements is not always possible and therefore some of the metrics measure more than one requirement. Each metric is evaluated on a scale from 0 to 1; where 1 signifies the agent perfectly meets the requirement. Section 5.1.1 describes how the set of requirement metrics can be combined into a requirements utility function that can be used to determine when and how to correct the interface agent's user model.

The metrics were chosen for the "insight" they provide for the interface agent's utility for offering assistance. The set of metrics is admittedly *ad-hoc*. This is not the only set of possible metrics. Empirical tests should be performed to validate the metrics used actually measure all aspects of the interface agent's behavior. However, these tests were out of scope for this dissertation. The usefulness of the metrics is argued on a philosophical basis. While this research focuses on illuminating the nature of what is required for an interface agent to provide beneficial assistance, it also provides a reasonable first step towards providing an empirical analysis of the requirements by providing well-reasoned justification for the use of a set of metrics.

This methodical way of measuring an agent's ability to provide assistance that can be incorporated into an interface agent architecture. Other researchers fail to provide an explicit means to measure the utility of their agent's assistance.

The following four requirements capture the essence of interface agents.

Definition 1 *Adaptivity* *— the ability to modify an internal representation of the environment as a result of sensing of the environment in order to change future sensing, acting, and reacting for the purpose of determining user intent and improving assistance.*

Adaptivity implies a number of sub-requirements. It implies an agent is *perceptive.* That is, the agent can distinguish relevant features in the environment in relationship to the method necessary to act and react within the environment. Concerning an agent's ability to act and react, the adaptivity requirement assumes the agent is able to affect the environment through its acting and reacting. *Reactive* "behavior" implies a timely response (i.e., stimulus-response) to sensed events, whereas to *act* connotes a deliberative (reasoned) response to events. The relationship between the agent's sensing, acting, and reacting with regards to the internal representation of the environment further defines properties for the agent. Goodwin (1993) defines these deliberative agent properties to include *predictive* — the ability to model the environment so as to predict how its actions will affect the environment; *interpretive* — the ability to correctly assess its sensors; and *rational* — the ability to perform actions to obtain its goals.

The *precision metric* measures the interface agent's ability to accurately suggest assistance to the user. The precision metric is defined as

$$M_{precision} \triangleq \frac{number\ of\ correct\ suggestions}{number\ of\ suggestions}. \tag{7}$$

Correct assistance includes *not* offering assistance as a result of the expected utility of the highest ranked goal and the user's chosen assistance thresholds.

The precision metric does not account for the agent's failure to suggest assistance when assistance should have been offered. For example, if the agent only makes one suggestion and that suggestion is correct, Equation (7) evaluates to 1. However, if the agent should have suggested assistance more often, but was unable to, the precision metric does not measure this problem. An inability to offer assistance arises from the following situation. The CIA architecture attempts to determine the type of assistance to offer the user every time an observation is communicated to the interface agent. If a new observation is communicated to the agent before it determines the type of assistance to offer (including offering no explicit assistance), the agent abandons attempting to offer assistance based on the previous observation and starts another Bayesian belief updating cycle based on the new observation. To measure this aspect, an assistance capability metric is defined, measuring the agent's *capability* to offer suggestions, as defined by the percentage of times an agent is able to suggest correct assistance based on a particular "state of the world" before the state of the world changes (e.g., the agent receives more information from the application about the environment).

This metric, in effect, measures the ability to provide correct assistance to the user and is defined as

$$M_{assistance\_capability} \triangleq \frac{number\ of\ correct\ suggestions}{number\ of\ state-of-world\ changes}. \tag{8}$$

A slight revision of this metric, called the *perceptive* metric, measures the interface agent's ability to perceive the state-of-the-world change and offer assistance, whether it is correct or not. The main reason for having this metric is because a suggestion is considered correct only if the user explicitly indicates the assistance is correct or conversely indicates the assistance is incorrect. The user may only explicitly indicate the suggestion's correctness if the interface agent makes a collaborative suggestion. This conservative approach to assessing whether a suggestion is correct ensures the CIA architecture does not wrongly assume a suggestion was correct. The perceptive

metric is defined as

$$M_{perceptive} \triangleq \frac{number\ of\ suggestions}{number\ of\ state-of-world\ changes}. \tag{9}$$

The precision and assistance capability metrics are related to the precision and recall statistics, respectively, used in text filtering systems (Oard and Marchionini 1996); where the precision statistic is defined as a ratio of the number of relevant found documents to the total number of documents found and the recall statistic is defined as the fraction of the actual set of relevant documents that are correctly classified as relevant. As an example, a value of 1 for $M_{precision}$ means the interface agent always makes correct suggestions when capable.

The *reactive metric* measures how quickly the agent can respond (i.e., act) to an environmental stimulus. We define $T_{sense \to act}$ as the measure from the time the stimulus is sensed (i.e., received) by the agent to the time the agent is able to act. Note that no explicit action is considered an action. For example, the agent may determine the user does not need assistance currently. To ensure $M_{reactive} \in [0,1]$ a user-defined cutoff time, T_{cutoff}, is provided. The agent must provide assistance within this time limit otherwise the interface agent will not suggest assistance. This metric is defined as

$$M_{reactive} \triangleq 1 - \frac{T_{sense \to act}}{T_{cutoff}} \tag{10}$$

As an example, if the interface agent is unable to respond within T_{cutoff}, $T_{sense \to act} = T_{cutoff}$ (by design) and therefore $M_{reactive} = 0$.

Definition 2 Autonomy — *the ability to sense, act, and react over time within an environment without direct intervention.*

This requirement, more than any other, seems to define agency as Petrie (1996) argues. However, Petrie also notes autonomy is not well defined within the community. Franklin and Graesser (1996) argue autonomy implies a reactive (sensing and acting within a time constraint), temporally continuous, and goal-oriented (pro-active)

agent. Direct intervention means explicit "activation" by the user or other agents. This definition does not preclude our agent from responding to this sort of collaborative interaction with the user and other agents. Instead, it is desired that the interface agent be able to act on behalf of the user without being "told" to do so. Indirect intervention takes the form of "looking over the shoulder" (Maes 1994a) of the user to determine what the user is doing and determining how to assist the user. Autonomy implies some sort of saved internal state, whereas adaptivity explicitly requires it. One external metric measures the number of suggestions the agent offers to the user without the user's request. This metric is defined as follows:

$$M_{external\_autonomy} \triangleq \frac{number\ of\ autonomous\ suggestions}{number\ of\ suggestions}. \tag{11}$$

A value of 1 for $M_{external\_autonomy}$ means none of the agent's suggestions are the result of the user explicitly requesting help from the agent.

Definition 3 _Collaboration_ — _the ability to communicate with other agents, including the user, to pursue the goal of offering assistance to the user._

All interface agents collaborate with users. Collaboration with the user best differentiates interface agents from other types of agents. This collaboration may be as simple as making a suggestion to the user and asking if the suggestion was correct or not, or as complicated as observing the user's actions within the environment, attempting to determine the needs and intent of the user, and providing assistance at "appropriate" times. Collaboration allows agents to increase their internal representation accuracy, resolve conflicts and inconsistencies within the representation, and improve their decision support capabilities (Sycara et al. 1996). Collaboration may also involve non-human agents. For heterogeneous agents, collaboration implies an agreed upon agent communication language and a commitment to use that language. Central to collaboration are the _behaviors_ an agent can perform and the _protocol_ in which they communicate those behaviors (Decker et al. 1997).

Collaboration metrics are therefore used to measure both collaboration with the user and other agents — in the case of this dissertation, correction adaptation agents (discussed in Section 5.3). With regards to measuring the collaboration requirement with correction adaptation agents, we are concerned with the efficiency of the collaboration using the agent's communication language[1]. There is a plethora of performance metrics for communication protocols (Spragins, Hammond, and Pawlikowski 1991) that we can use to determine the ability to communicate with other agents over networks. While not all the factors determining these metrics are within the control of the interface agent (e.g., network throughput is largely determined by the bit rate), knowledge of communication bottlenecks can help the interface agent make informed decisions about accessible and applicable information sources. With regards to measuring the interface agent's ability to collaborate with the user, we are concerned with the effectiveness of the collaboration. We define a metric to measure the level of collaboration. First, we define a metric to measure the percentage of the "workload" the agent performs versus the user as follows:

$$Agent_{workload} \triangleq \frac{\# \ of \ agent \ actions}{\# \ of \ agent \ actions + \# \ of \ user \ actions}. \qquad (12)$$

Note that this metric is based on actions and not suggestions since a suggestion may contain several actions. Each user can determine the amount of collaboration he/she desires by setting a *collaboration threshold*, ranging from 0 to 1, specifying the desired amount of collaboration between the interface agent and user. For example, a collaboration threshold value of 0.5 means the user desires the "workload" to be divided evenly between the agent and user, whereas a threshold of 0 indicates the user wants the agent to perform no actions on his/her behalf. Using this threshold, we can define the *collaboration metric* as follows:

$$M_{collaboration} \triangleq 1 - |T_C - Agent_{workload}|, \qquad (13)$$

---

[1]We are currently using KQML (Mayfield, Labrou, and Finin 1996).

where T_C is the user's collaboration threshold.

Definition 4 *Robustness* — *the ability to degrade assistance gracefully.*

Robustness is required for interface agents since they, more than other agents, must be capable of gracefully degraded performance because they must interact with the most complicated of agents — users. However, robustness is not limited to performance. Extensibility and maintainability are also key. It is desired for the agent to possess the ability to easily adapt to information and requirement changes. The former is best dealt with under the adaptivity requirement (e.g., the ability to adapt to different users), while the latter is better dealt with during agent specification. The agent should have the ability to add new sensor information dynamically or be used in a new environment with different requirements. The ability of the interface agent to correct its user model is partially related to the number of correction adaptation agent responses received. This correction process (described in detail in Section 5.3) relies on the interface agent sending a "bid" to the correction adaptation (CA) agents, and they in turn responding within a time limit, $T_{cutoff_{CA}}$. Assuming N_{CA} correction adaptation agents, the metric is defined as follows:

$$M_{response\_quantity} \triangleq \frac{number\ of\ CA\ agents\ responding}{N_{CA}}. \tag{14}$$

Similarly, we measure the average time it takes correction adaptation agents to respond:

$$M_{response\_time} \triangleq 1 - \frac{\dfrac{\sum\limits^{N_{CA}} T_{CA_i}}{N_{CA}}}{T_{cutoff_{CA}}}. \tag{15}$$

5.1.1 Requirements Utility Function. A successful agent is defined as "an agent with the ability to provide timely, beneficial assistance (suggestions, tutoring, help, interface adaptations)." This definition appears very open ended. The follow-

ing discussion attempts to close the ends by looking at the agent's utility in meeting these requirements.

Section 2.3 discussed a number of machine learning techniques for eliciting knowledge for use in user models. In addition to elicitation of new knowledge, machine learning techniques are useful for maintaining user model knowledge. Höök (1997) states interface agent designers need a better understanding of how intelligence can *improve* the interaction with the user. This understanding includes *how* machine learning techniques affect the user model and therefore interaction between the interface agent and user. Failure to understand this relationship may result in a fruitless search for ways to improve the user model.

The set of requirement metrics allows the interface agent to readily identify which requirements are not being met. These metrics can be combined to obtain an overall qualitative measure of how well the interface agent meets its requirements.

A utility function $U_{requirements}$ is defined for the requirement metrics of the agent, weighted with respect to their importance, based on some previous history. That is,

$$U_{requirements} : \omega^n \times R^n \times H \mapsto \Re,\qquad(16)$$

where for each history $h \in H$ of previous actions and events, $\omega \in [0,1]$ is a weighting factor for each of the n requirement metrics R (e.g., $M_{response\_time}$), and the utility function maps to a real number. The weight ω can be a function of time, where the weights are allowed to change depending on the current situation. For example, if the interface agent is making poor suggestions, the weight(s) associated with the adaptivity requirement metrics can be increased, denoting its increased importance. The higher the value of the utility of our interface agent, the more "successful" it is in meeting its requirements.

The interface agent will not make suggestions if the requirements utility function falls below the user-defined threshold. That is, the interface agent "knows," as

81

determined by the requirements utility function, when the user model is not accurate enough to make suggestions. The interface agent continues to observe the user's actions in a "sit-and-watch" mode. Furthermore, if the requirements utility function improves, the interface agent will make suggestions again.

The metrics do *not* explicitly take into account the recency of the actions determining the metrics. As presented, each metric is calculated over the entire history of actions and events. However, in practice, it may be desirable to have a fading function to weigh more heavily recent actions/events. Note, however, the utility function $U_{requirements}$ does take history into account. If we desire to evaluate the metric for the last ten events, for example, the interface agent limits the history to the last ten events and evaluate the metrics over this history. It may also be desirable to have metrics evaluated over several different history lengths[2].

5.2 User Model Problems

There are several types of problems that may arise in an interface agent's knowledge representation (Brown, Santos Jr., and Banks 1997; Banks, Stytz, Santos Jr., and Brown 1997), as related to the agent's *relevancy set* (Brown, Harrington, Santos Jr., and Banks 1997). These problems include the following:

- **Absolute thrashing** — an observable property (i.e., node) repeatedly entering and leaving the relevancy set. The problem results from adding a node to the user model only to have it never become part of a relevancy neighborhood, and be removed from the user model some time later. The node's actual relevancy must be brought into question.

- **User thrashing** — occurs when a user's intent radically changes from one extreme to another. If we are truly modeling user intent, we should desire to capture user thrashing. However, we also desire to avoid thrashing of the

---

[2]The advantage to evaluating metrics over varying history lengths is akin to stating mutual fund performance over 5, 10, and 20 years performance.

system's measure of user intent so we can make accurate predictions of the user's intent. As a concrete example, consider an aunt who is known to have drastic mood swings. You are aware of this, and develop ways of observing her current behavior to find a promising way of approaching her. You never vary your approach drastically.

- **Class thrashing** is the result of a user who is diametrically opposed to his/her user class and therefore the network initially does not represent a user well and consequently, must "learn back" a user's behavior. This type of thrashing not only affects the user by making incorrect inferences, but affects the user class the misplaced user is currently a member of.

Related to the above problems are the concepts of *rate of divergence* and *rate of convergence*. The rate of divergence is a measure of how quickly a node leaves the user model. We are concerned with ensuring a node that was added to the user model, but is not used often, will exit the network quickly, therefore allowing more useful nodes in the relevancy set. The rate of convergence is a measure of the speed the "momentum" of past observed behavior is overcome by changes in current behavior and therefore, how quickly the expected utility of a node will settle out to a particular value is important in conjunction with class thrashing. We desire to know how fast a user model will allow a user to overcome past behavior. For example, a user may exhibit a particular behavior for a "long" time and then suddenly change behavior, perhaps as the result of some new stimuli in the user's environment. A fast rate of convergence will quickly allow the user model to overcome the past behavior and accurately model the current behavior.

These problems are a result of the interface agent failing to meet its requirements and indicate that the agent's user model is inadequate to deal with the dynamic environment. Since the interface agent possesses a set of requirement metrics and an associated requirements utility function, the agent can readily identify which

requirements are not being met and attempt to correct the problem by altering its user model.

5.3 Correction Model

This section addresses *when* and *how* to dynamically change the user model and prevent a quixotic search for ways to improve an inaccurate user model. The section discusses how a set of correction adaptation agents can use the aforementioned requirement metrics, requirements utility function, and previous interface agent behavior to suggest changes to the user model.

Every time new observations are received or periodically, the interface agent calculates the value of the requirements utility function to determine if the interface agent needs to correct the user model to more accurately reflect the user's intent. The implemented approach to correcting the user model is to have the interface agent request "help" from correction adaptation agents — special agents capable of correcting problems with the user model by adapting it to improve the interface agent's requirements utility. This section discusses a scenario for having a correction adaptation agent suggest changes to the ailing interface agent user model, based on the concept of a contractual bidding process. These agents engage in a "bidding process" to recommend changes to the ailing interface agent user model. The interface agent serves as a *manager* agent, responsible for determining when a contract is available (in this case, when the requirements utility function value is below a user-defined threshold), announcing the contract to be filled, receiving bids from the *bidder* agents (i.e., correction adaptation agents), and finally accepting or rejecting the bids based on their utility. Correction adaptation agents are responsible for altering the user model based on their specific bidding behavior component, evaluating the *utility* of the alteration, and making the bid. The correction adaptation agent that may improve the interface agent's requirements utility the most "wins" the contract to correct the user model.

84

In general, correction adaptation agents have *no* explicit domain knowledge, only user model knowledge (e.g., the user model parameters)[3]. They are "concerned" with increasing the utility of the user model. However, this lack of domain knowledge is an asset. We can use them for different Bayesian network user models regardless of the domain of the target application. However, as mentioned previously, certain correction adaptation agents may be more or less useful for certain domains due to the sort of interactions a user may have with the application. Therefore, we are even more concerned with defining the "right" correction adaptation agents. The idea of building the right correction adaptation agents is investigated in Section 5.5. The correction adaptation agents implemented for this dissertation are described in Appendix D. These agents are responsible for altering user model profile parameters, learning the Bayesian network user model conditional probabilities, and performing knowledge acquisition.

Two points need to be addressed. First, the use of correction adaptation agents to suggest adaptations to the interface agent's user model may not seem intuitive. It may appear the interface agent itself is better equipped to correct its own user model. However, as mentioned previously, our agent already possesses a number of ways to adapt to the dynamic environment. The situation where the agent's requirements utility falls below the threshold indicates that its adaptation mechanisms are incapable of dealing with the dynamic environment and needs specialized "attention." Secondly, the use of correction adaptation agents allows the interface agent to continue to observe the environment and possibly still offer assistance without utilizing computational resources correcting the user model.

The bidding process model used is adopted from Müller (1996) and is used as the underlying meta-level learning mechanism for the interface agent. The model is formally defined as follows:

---

[3]This restriction is relaxed. See Appendix D.

Definition 5 *Let D represent the domain. The model can then be represented as a 5-tuple (A, N, U, Π, Σ) where*

- $A = \{a_1, \ldots, a_n\}$, $n \geq 2$, *is a set of agents a_i with mental states $B_i = (O, h, E)$, where*

    - $O \subseteq D$ *is a domain ontology.*

    - $h \in H$ *is a history of interface agent suggestions, any information the agent used to make its suggestions, and user choices based on the assistance offered.*

    - E *is the effectiveness of the agent, with respect to the various requirement metrics.*

- $N \subseteq D$ *is the negotiation set and semantically represents the changes a bidder agent proposes to make to the user model.*

- $U = \{u_1, \ldots, u_n\}$, *where $u_i : A \mapsto \Re$ is a utility function for correction adaptation agent a_i, based on the interface agent's requirements utility function over time[4].*

- $\Pi = (K, \pi)$ *is a negotiation protocol, where*

    - $K = \{start, done, ANNOUNCE, BID, REJECT, ACCEPT, REPORT\}$ *represents the communication primitives.*

    - $\pi : A \times K \mapsto 2^K$ *is a protocol function mapping communication primitives for agents to allowable reactions.*

- $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ *is a set of negotiation strategies where $\sigma_i : \Pi \times A \times K \times 2^D \times U \mapsto K \times N$. Specifically, $\sigma_i(\Pi, a_i, k, N, u_i) = (k', N')$ with $k' \in \pi(a_i, k)$, $N' \subseteq N$.*

The correction adaptation agents' architecture is very similar to the interface agent's architecture (Figure 4), where each correction adaptation agent possesses

---

[4] *As mentioned previously, the weights for each requirement metric used to calculate the requirements utility function may change over time, and therefore, the weights are stored in the history H for use in calculating the correction adaptation agents' utility over the history.*

its own user model, an evaluator for calculating the metrics and requirements utility function, and a communication protocol handler for communicating with other agents. Additionally, each correction adaptation agent has a specialized bidding behavior component capable of adapting the user model. In practice, each correction adaptation agent maintains a user model that is identical to the interface agent's user model until the interface agent requests help from the correction adaptation agents, at which time each correction adaptation agent adapts its own user model based on its bidding behavior component. This engineering design decision was the result of fear of communication latency problems with communicating large user models via KQML messages.

The actual detailed definition of Π and Σ is more appropriately left for an appendix (see Appendix C). Any number of bid strategies can be adopted for the CIA architecture. Currently, the negotiation protocol function, Π, and negotiation strategy, Σ, are defined generally the same as Müller (Müller 1996), which is a sealed-bid, single award strategy. That is, the other agents have no idea what the "price" (i.e., utility) other agents are bidding. Other strategies include agents learning from past bids to improve future bids (Zeng and Sycara 1996).

5.4 Performing "What If..." Analysis

The negotiation set defined for the correction model above is concerned with *what if* changes were made to the user model. Specifically, if the interface agent had made a change to the user model (e.g., the fading function for a goal was changed, actions are added and/or removed from a goal, a new goal is created, etc.) at some previous timeslice, would the change have improved the user model? The correction adaptation agent evaluates the utility of the proposed change(s) by making the changes to the "oldest" user model stored in the history H. Then, for each suggestion made to the user, the correction adaptation agent determines what the new suggestion(s) would be, based on the new user model and any evidence stored in the

87

history. The requirements utility function value is then recalculated at each time slice, up to the current time slice. The "winning" correction adaptation agent is the agent improving the requirement utility function by the greatest magnitude. This evaluation can be biased by multiplying the bidder agent's effectiveness $E \in [0, 1]$ and the requirements utility function. The correction adaptation agents' effectiveness E is updated by simple reinforcement learning, where the "winning" agent receives positive learning. The reinforcement learning takes into account those bidder agents determined to be "helpful." In practice, certain correction adaptation agents are more apt to improve certain metrics than other metrics. That is, the correction adaptation agent's bidding behavior component affects only certain metrics. Therefore, if a particular metric has a significantly greater weight, ω, than the other metrics and/or if this metric's value is significantly lower than the other metrics, the interface agent is likely to choose those correction adaptation agents capable of increasing that metric's value, thus increasing the requirements utility function's value.

The philosophy behind the correction model deserves some attention. The correction agents cannot change the past. User actions, environmental stimuli, etc. are fixed in the past and cannot be changed. What can be changed is the knowledge the interface agents possesses in making its decisions and therefore, possibly the future assistance offered to the user. This knowledge is obviously stored in the user model. If the set of requirements metrics is considered, there are certain metrics that will be difficult to recalculate over the history. For example, $M_{precision}$ in Equation (7) depends on the number of correct suggestions. The *only* way the interface agent can determine it has absolutely made the correct suggestion is to ask the user[5]. Since the correction adaptation agents recalculate over history, the user cannot tell

---

[5]Note that this determination means the interface agent does not know absolutely whether autonomous assistance is beneficial.

the interface agent whether a suggestion is correct or not. Some metrics, such as $M_{reactive}$ in Equation (10), are easily recalculated.

Recall that Equation (4) allows the interface agent to obtain a rank ordering over all possible assistance. This ranking, stored in the evaluator's history stack in each timeslice, can be used by the correction adaptation agents to perform a "what if" analysis. Based on this discussion, another requirements metric can be defined. The *misconception* metric is defined as follows:

$$M_{misconception} \triangleq 1 - \frac{getGoalRanking(\alpha)}{number\ of\ goals},\ \ (17)$$

where $getGoalRanking(\alpha)$ computes the goal ranking between 1 and the total number of goals of the highest ranked goal associated with the action α. A metric value of 1 indicates no misconception exists; on the other hand, a value of 0 indicates a massive misconception.

The misconception metric is aptly named for the function it is intended to serve — identifying misconceptions in the system (to include the user). Intelligent tutoring systems (ITS) are particularly concerned with determining when a user has a misconception about a concept. For example, ANDES (Conati et al. 1997; Gertner et al. 1998) detects when a user does not know a physics concept or incorrectly applies a concept. Misconceptions arise from two different areas. If the interface agent could determine if the user is pursuing a goal, or should be pursuing a goal, but the user is performing actions unrelated to the goal, it is possible the user either (1) is unaware of the actions required to achieve a goal or (2) considers another goal to be more important. The former is an indication of a user misconception. The latter is indicative of an inaccurate user model, which in essence is an interface agent misconception. Specifically, the utility of the goal has been over estimated

with regards to the goal the user feels is more important[6]. Note that ITS typically assume the domain knowledge is accurate and therefore can rule out the latter. The CIA architecture does not make this assumption.

5.5 Correction Adaptation Agent Evolution

The motivation of this section is driven by the simple fact that constructing correction adaptation agents "by hand" is a time consuming task. Interface agent designers could attempt to build several (many) agents capable of different adaptations. However, we must ask the following questions: Which ones do we build? What is really needed?

Initial research into interface agents identified problems that may occur in the user model (Brown, Santos Jr., and Banks 1997). Although it is hypothetically possible to construct more agents to repair the problems identified, a better understanding about why certain correction adaptation agents are better suited to correcting certain metrics is needed. A top-down approach to the construction of correction adaptation agents allows a decomposition of the correction adaptation agents into atomic parts, determining first how the agents are composed. Given this decomposition, it is then desired to be able to combine the various parts together to make new agents.

For this discussion, domain independence is assumed. The correction adaptation agents are abstracted to words in a language. "Language" does not mean agent programming languages nor speech act theory languages. Instead, the term "language" is used in the computational theory sense. Each correction adaptation agent is modeled as a word in the language produced by a grammar.

---

[6]It is possible the Bayesian network user model is incorrect. For example, an action may be incorrectly associated causally with a goal. While although this can also be considered an interface agent misconception, this discussion does not assume this sort of user model inaccuracy.

5.5.1 User Model Construction Heuristics. When a user model designer attempts to determine what is important in the domain to model, he/she uses one or more heuristics, or general rules of thumb, to guide his/her knowledge acquisition task. These heuristics attempt to capture salient characteristics that define user intent by discriminating certain observations as being relevant to determining the user's intent. These heuristics are not mutually exclusive and designers of user models typically use more than one heuristic when designing the user model. As such, we must identify the common characteristics of user model elicitation and adaptation techniques. Interface agents adapt their user models as a result of applying one or more heuristics. Relevant heuristics identified during this research are as follows:

- **Causality** — *Why* a user performs actions. The user intent ascription philosophy presented in Chapter III states that a user perform actions in response to environmental stimuli and to achieve some goal. For example, Jameson (1996) describes how a causal planning model can be used to construct Bayesian networks.

- **Context** — *What* is the current context. For certain types of interface agents — most notably agents for information filtering and/or data mining — the current context is important. Taking an example from Goldman and Charniak (1993), if a user is referring to a bank, it is useful to know whether he/she are referring to a financial institution or a river bank. The context of previous interactions may help disambiguate the current use of a word.

- **Frequency** — *How often* a user performs an action. Some interface agents use a fading function so actions become less relevant as time progresses.

- **Human-Factors** — *Who* is the user. Knowing user information *a priori* can be useful for adapting the interface to the user's needs. Human-factors such as psychological factors (e.g., spatial ability, cognitive ability, temporal ability), as well as physiological factors (e.g., skill level, age) may be directly applicable to the user's needs.

91

- **Modality** — *What* modes a user prefers, or uses explicitly or implicitly. This heuristic captures a large portion of the meaningful characteristics in direct manipulation interfaces. For example, what skill level (expert, intermediate, novice) does the user prefer? What type of ways do they like to view their information (e.g., full page, page layout, outline)? What about presentation methods such as textual, graphical, or audible? Do they prefer natural language or not? The specific "tools"[7] a user prefers to use can be considered a mode of the application. The orientation, size, position, etc. of the various windows, icons, and menus can also be considered a mode.

- **Resource Usage** — *What* resources a user needs. A resource can be as simple as a file or printer, or it can be another application, such as a World Wide Web search engine. As an example, Yoshida (1997) predicts Unix resource prediction using graph-based induction.

- **Temporality** — *When* a user performs an action. Do they perform a sequence of actions upon starting the application or prior to exiting? Does a user always react/respond a particular way to a certain action? The actions may be executed by the interface agent on behalf of the user.

Any of the heuristics presented here may be combined. However, determining which heuristics work "well" together is a difficult process and one not typically addressed by researchers. It is essential to measure the effectiveness of the user model constructed using these heuristics against a set of measurable requirements. This research has already presented a methodical way of doing this in Section 5.1. Determining the granularity used to model the domain is also important. For example, is it important to know the user activated the spell checker via a pull-down menu versus a hot-key?

---

[7]Tools are defined to mean any function that helps the user get his/her job done. A tool can be a spell checker in a word processor, a macro, or a meta-method of activating another tool, such as using hot-keys, pull-down menus, and/or icons.

5.5.2 The Correction Grammar and Language. This section introduces a proposal for a context-free grammar (CFG) generating a language of agents for behavior maintenance. The basis of the grammar is the agent definition given in Section 2.1. The context-free grammar is defined using Backus-Naur Form (BNF) as follows:

$$
\begin{aligned}
< \text{agent} > \ &\triangleq\ < \text{sense} >< \text{think} >< \text{affect} > \\
< \text{sense} > \ &\triangleq\ < \text{read sensors} >< \text{transform raw sensor data into} \\
&\qquad \text{internal knowledge representation} > \\
< \text{think} > \ &\triangleq\ < \text{inference over knowledge representation} > \\
&\qquad < \text{compute confidence} > \\
< \text{affect} > \ &\triangleq\ < NOP >\ |\ < \text{internal affect} >< \text{external affect} > \\
< \text{internal affect} > \ &\triangleq\ < NOP >\ |\ < \text{write new state} >< \text{internal affect} > \\
< \text{external affect} > \ &\triangleq\ < NOP >\ |\ < \text{physical affect} >< \text{external affect} >
\end{aligned}
$$

For the CIA architecture presented in this dissertation, sensing involves observing environmental stimuli and user actions (reading sensors) and affecting involves offering suggestions (external affecting). The decision theoretic approach taken in this dissertation is implemented by an agent's "think" rule. "Thinking" involves inferencing over the Bayesian network user model knowledge representation. Thinking involves utilizing the current sensor data as represented in the knowledge representation and may or may not involve using past states of the world (i.e., history) as evidence for the Bayesian belief updating. The CIA architecture uses the expected utility as its "confidence factor." The agent's *internal_affect* provides a mechanism for modifying previous observations in light of new ones. Furthermore, the agent's correction model is a way of modifying the internal representation of the world state.

93

In addition to the preceding rules, meta-rules for constructing rules can be defined. These meta-rules are similar to the meta-rules presented by Goldman and Charniak (1993).

$$< \text{meta-function} > \quad \triangleq \quad \rightarrow < \text{sensor proposition} >$$

$$\text{if} < \text{sensor name} > = < \text{sensor value} >$$

$$< \text{true action} > < \text{false action} >$$

$$< \text{sensor-proposition} > \quad \triangleq \quad < \text{sensor name} > < \text{sensor value} >$$

The concept of meta-rules is to provide the interface agent the ability to determine which rules should be created, what pieces should be combined and which ones should not be, what works well together and what does not. For example, a probabilistic confidence factor for if-then rule inferencing may not work well. It is desirable to use some metric/discriminator in determining how to form rules. The discussion in the next section addresses this desire.

5.5.3   Correction Adaptation Agent Performance Metrics.   For any given set of agents generated by a grammar, we are concerned with whether or not we have the right "mix" of agents. Choosing the wrong correction adaptation agents will result in a set of agents possibly incapable of correcting an inaccurate user model. Also, given a grammar, which production rules produce the best language for correcting a specific interface agent's user model? That is, which rules generate a language that gives us the best performance?

Balch (1997a) presents the concept of *behavioral diversity* for a multi-agent robot soccer system and presents experimental results showing that more diverse teams perform better (e.g., score more points). His behavioral diversity is measured by a social entropy metric (Balch 1997b). He states the following desirable properties of his social entropy metric:

- The least diverse society is one in which all agents are equivalent. Conversely, the most diverse society is one in which no agent is equivalent.

- A society where one agent is different and all other agents are equivalent is more diverse than one where all agents are equivalent.

- If two societies have uniformly-sized groups (where a group is a collection of equivalent agents), the one with more groups is more diverse.

The crux of Balch's research is determining the benefit of heterogeneous team behaviors in the performance of a cooperative task. His research, as well as others (Goldberg and Matarić 1997; Haynes et al. 1995), indicates behavioral diversity is beneficial in some tasks. However, robot soccer, as well as hunter-prey scenarios used by some of the other researchers, are toy domains. His robots can only display one of three possible behaviors (move to ball, get behind ball, move to back field) for one of two situations (behind the ball or not behind the ball). For teams of four players, there are less than 7000 possible combinations. The domains used in this research are much more complex.

Any performance metrics defined for the CFG presented above should capture Balch's social entropy metric properties also. These properties have the following effects with regards to the CFG: as words are eliminated from the language, the diversity metric should decrease. Alternatively, a language with a higher diversity metric should be able to correct more problems that might occur with the user model. The evolutionary computation community has studied the idea of diversity, and the related topic of convergence. In general, for a given population, diversity is desired to prevent premature convergence on a local optimum.

To measure how diverse a set of correction adaptation agents are, consider a perfect grammar, capable of generating all possible correction adaptation agents. Denote this grammar G and the language of correction adaptation agents generated by G as language L.

Lemma 1 *Let G and G' represent the grammars of two agents, respectively. Let L and L' be the languages generated by G and G', respectively. Given $G' \subset G$, then $L' \subset L$.*

That is, G' can generate only a portion of the correction adaptation agents compared with G. To measure this, a coverage metric is defined as:

$$M_{coverage} \triangleq \frac{|L'|}{|L|}. \tag{18}$$

Since $L' \subset L$, $M_{coverage} \in [0, 1]$. Obviously, $M_{coverage} \approx 1$ is desired.

The above definition assumes finite languages. What if L is infinite? There are several possible solutions:

1. Restrict the format of production rules of grammar. If no directly recursive rules are allowed (e.g., $S \to aS$) or indirectly recursive (e.g., $S \to aA; A \to bS$), L will be finite. However, this restriction may be severely limiting and uninteresting.

2. Change the metric definition. The number of rules in the grammar is finite. The coverage metric is redefined as follows

$$M_{coverage} \triangleq \frac{|G'|}{|G|}. \tag{19}$$

   This metric definition begs the following question: if two grammars have the same value for the coverage metric, are they equal? That is, can they correct the same types of problems that might occur?

3. Provide an equivalence class of agents. If the coverage metric is defined over the total number of equivalence classes of words (i.e., correction adaptation agents), a finite number of equivalence classes is guaranteed. This raises the interesting question: do certain agents subsume others? However, to define this coverage metric based on equivalence classes, we must define the equivalence operator.

96

One suggestion is to "group" words (i.e., correction adaptation agents) by the metrics they can improve.

Two correction adaptation agents (words in the language) may provide similar correction adaptations. Under resource constraints, the inclusion of both correction adaptation agents in the collection of agents the interface agent considers "bidders" may be harmful. Therefore, a method of determining the *individualism* of a correction agent is warranted.

The following definition is presented to aid in the development of an *individualism* metric.

Definition 6 *Let* $\rho_i^{CA_j}$ *be the* capability *of correction adaptation agent j to improve metric* ρ_i. *This capability is defined as the percentage of bids the* CA_j *was able to improve metric* ρ_i.

Given this definition, the individualism metric is defined as

$$ D(CA_i, CA_j) \triangleq M_{individualism} \triangleq \frac{\sum_{i=1}^{n} |\rho_i^{CA_i} - \rho_i^{CA_j}|}{n}, \tag{20} $$

where n is the number of requirement metrics as given in Equation (16). This defines how different a correction adaptation agent (word) is from another.

Balch (1997a) proceeds to further define a social entropy metric based on Information Theory (Blahut 1987) that determines the heterogeneity (i.e., diversity) of the entire collection of agents, taken as a group. He defines the concept of *castes* of agents. An agent belongs to a caste if its individualism metric differs from other members of the caste by no more than some user-defined ϵ. He terms this concept "ϵ-equivalence." Given this discussion, the following two equations from Balch (1997a) define social entropy for a group G of agents:

$$ p_i = \frac{|C_i|}{\sum_{j=1}^{c} |C_j|} \tag{21} $$

97

$$Het(\mathcal{G}) = -\sum_{i=1}^{c} p_i \log_2(p_i) \qquad (22)$$

where C_i is a caste. The values p_i are normalized so that $\sum p_i = 1$ since an agent may belong to more than one caste.

Why do we care about diversity? In a perfect world, the use of grammar G to generate every possible word allows the CIA architecture to handle any problem. However, we do not live in an idealistic world. We therefore desire to have the "best" subset of words in L. It is conjectured that determining *a priori* those words giving the most diverse group of words will yield the best results. Over time, the best set of words is that set of words determined to be probabilistically useful.

Conceptually, $M_{coverage} = 1$ (Equation (18)) indicates the language L' generated by G' contains all possible words. As already stated, this case is unrealistic. This metric definition begs the following question: If two languages have the same value for the coverage metric, are they equal? That is, can they correct the same types of problems that might occur? The metric as defined does not answer these questions. However, based on the discussion of social entropy, what is truly important is to ensure the right castes exist to correct user model inaccuracy problems. Membership in a particular caste implies all members of that caste are capable of improving the values of the same metrics. If all castes are considered, does there exist at least one caste capable of correcting each requirement metric? Based on this discussion, the following metric measures metric coverage:

$$M_{metric\ coverage} \triangleq \frac{number\ of\ metrics\ covered}{n} \qquad (23)$$

where a metric is considered "covered" if there exists a caste capable of improving that metric and n is the number of metrics.

The grammar, language, and performance metrics can be used to dynamically define a set of correction adaptation agents capable of correcting an inaccurate user

model. The use of the language and metrics allows the interface agent to *a priori* compute which words (i.e., correction adaptation agents) are best suited for the situation. If $M_{metric\ coverage} = 1$, the interface agent can use the classic *set-covering problem* algorithm (Cormen, Leiserson, and Rivest 1990) to determine the smallest set of correction adaptation agents capable of improving all the requirement metrics. However, since the language is possibly infinite, we desire to limit the size of the language to the assumed best agents. If this set of "best" agents turns out to be a collection of poor performers, the interface agent can add/delete agents based on their performance.

5.6 Conclusion

Accurate user models are necessary for ascription of user intent. This chapter investigated a systematic way of insuring the CIA architecture user model is always accurate. Four requirements for interface agents were represented. The interface agent's ability to meet these requirements is measured by requirement metrics and a requirements utility function. The use of this utility function performs two main functions within the CIA architecture. First, it determines *when* a user model is inaccurate. Secondly, it determines *how* to fix the user model.

The use of a multi-agent system of correction adaptation agents allows the interface agent to dynamically determine the best possible correction of the user model. Each correction adaptation agent possesses a (possibly) unique method for correcting the interface agent's user model. The agents are domain independent and therefore can be used across different domains. The theory for correction adaptation agent evolution provides the catalyst for further study into agent team diversity. The difficulty in predicting the usefulness of certain correction adaptation agents and the desire for a more methodical design process for the construction of the correction adaptation agents led to the proposal presented in this chapter. The thoughts presented in Section 5.5 are extensible to the multi-agent systems research

99

field. If we consider, for example, each "player" in robot soccer as a correction adaptation agent, we can concern ourselves with which team players are best for a given scenario.

Although the simple grammar presented was not used in this research, future work proposed in Chapter VIII presents an idea of how the grammar might be utilized in the CIA architecture.

VI. Interface Agent Development Environment Architecture

The main problem with existing agent development environments (such as the ones presented in Section 2.2) is that they fail to be more than non-agent code development tools with a means of specifying a communication protocol. Most development environments focus on the communication aspect and distribution of a task to multiple agents. In general, one strength of agents is the ability to distribute tasks to specialized agents for handling those tasks (e.g., e-mail agents, calendar agents, information retrieval agents); focusing just on the communication aspects ignores other aspects of agent specification and development such as adaptivity and robustness. Furthermore, existing agent development environments treat the agent as the *whole* system. Interface agents are part of the whole system, and as such, much take into account the interaction between the user and target system and the interface agent, as well as any other agents (e.g., correction adaptation agents). The systems described in Section 2.2 suffer from at least one of the following weaknesses — lack of environment specification, adaptivity mechanisms, and agent knowledge representation and reasoning mechanisms. Additionally, these systems do not adequately deal with human-agent interaction. The underlying difference between agent-based and software engineering-based tools is the dynamics of the environment in which the agents must "perform," to include the dynamic needs of the user. This chapter, which is an expansion on the research presented by Brown, Santos Jr., Banks, and Stytz (1998a), addresses these issues explicitly by proposing development methods, tools, principles, etc. First, this chapter identifies the primary areas where agent development environments are deficient. Next, an agent specification language is proposed as a first step towards addressing the deficiencies. The approach used is contrasted with existing agent development environments, showing its strengths for support of interface agent development. This language supports the existing CIA

architecture by performing software engineering, knowledge engineering and acquisition needed to specify the interface agent's integration into a target system.

6.1 Agent Development Environment Deficiencies

Environment Specification. Existing environments implicitly address the environment where an agent must perform. An explicit representation of the environment can bring to the fore front critical domain considerations and enable the designer to identify those domain environmental features with which the interface agent must deal. These specifications can include system features (e.g., error messages, available resources, sensors, system events), user interface standards, as well as requirements and specifications required by the human interface developer, such as existing business practices and work flows. Additionally, an agent development environment must specify the target system's application programmer interface (API) that identifies the "marionette strings" the interface agent can "pull" to affect the target system (Metral 1993). Since this research is concerned with interface agents, it is desired to specify user behavior within the environment. This specification captures the relationship between the environmental stimuli (i.e., pre-conditions), user's goals, and actions to achieve those goals (i.e., the Bayesian network user model) as well as the user's profile and utility model. The use of an explicit specification of the domain environment for agent-based development is a technique used in software engineering (Rumbaugh, Blaha, Premerlani, Eddy, and Lorenson 1991).

Adaptivity Mechanisms. None of the environments explicitly address the fact that agents typically operate in dynamic environments, and therefore, must be capable of adaptive behavior. While adaptivity requirements can be addressed at the code level, they are better addressed at the specification level, where designers can determine what to adapt, how to adapt it, when to adapt it, and why. These adaptations must be in compliance with the environment specification.

Agent Knowledge Base and Reasoning Mechanisms. All but IBM's Agent Building Environment (IBM 1997), Reticular's AgentBuilder (Reticular Systems 1998), and the Agent Building Shell (Barbuceanu and Fox 1996a) fail to provide a persistent knowledge base and underlying reasoning mechanism for agents to reason about their environment via sensing, acting, and reacting. The other systems merely provide a way for the agent to react to an explicit activation; autonomous agent behavior, where the agent can reason about the actions of the user in an attempt to autonomously act on the user's behalf, is not addressed. A knowledge-based systems approach allows interface agent designers to separate the knowledge and reasoning mechanisms. This separation has two effects. First, a knowledge-based systems approach supports rapid prototyping of interface agent systems by allowing new knowledge to be incrementally added to the knowledge base as it becomes available. Second, the separation of the knowledge base and reasoning mechanism allows the reuse of the reasoning mechanism in other domains. A knowledge base also supports the persistent storage of a user model and allows the user model to be tailored to individual users.

6.2 Agent Specification Language

To address the short-comings of current agent development environments, the Agent Specification LANguage (ASLAN) is proposed. ASLAN shares some goals with existing knowledge-based software engineering specification acquisition tools (Lowry and Duran , pp. 292-302). In particular, ASLAN has the following goals:

1. To formalize the artifacts of interface agent development and the software engineering, knowledge engineering and acquisition activities, to include methodologies, principles, heuristics, that produce these artifacts. Formal specification languages enable specifications to be stated precisely and unambiguously. Formalization also facilitate the sharing and reuse of formally represented knowl-

103

edge. A specification of a representational vocabulary for a shared domain of discourse — definitions of classes, relations, functions, and other objects — is called an ontology. Gruber (1993) defines an ontology as a "specification of a conceptualization."

2. To assist with the development and validation of specifications. Development assistance helps users resolve conflicting requirements, refine incomplete and informal requirements, and use domain knowledge in developing the specifications. Validation methods can use the formal specification itself as a *prototype* or perform various types of analysis on the formal specifications.

3. To synthesize interface agent source code and user models from formal specifications. This approach enables maintenance to be performed by maintaining the specification, not the source code and user model.

ASLAN supports the existing CIA architecture and enables developers to easily perform the needed initial knowledge acquisition for the CIA architecture. The proposal extends, where possible, existing methods and tools, expanding their capabilities and coverage of agent specification and design issues. In some areas, new functionality is proposed; in others, the proposal merely extends existing functionality to account for adaptive interface agents. ASLAN is capable of incorporating business practices and user interface standards (e.g., the Defense Information Infrastructure (DII) standards for a Common Operating Environment(COE)) as an independent module that automatically drives agent interface developments. This methodology allows for automatic and "hands-free" updating of the ASLAN specification as business practices and user interface standards themselves are updated and changed.

ASLAN uses Ontolingua (Gruber 1993) with a knowledge acquisition tool front end. Ontolingua is a language for representing ontologies. In relationship to agent research, ontologies describe the concepts and relationships that can exist for an agent or a community of agents. Gruber (1993) uses ontologies for the purpose of enabling

104

knowledge sharing and reuse between agents. An ontological commitment is an agreement (by the agent) to use a vocabulary (i.e., ask queries and make assertions) in a way that is consistent with respect to the theory specified by an ontology. While Ontolingua is good for representing and sharing knowledge, it is poor for acquiring that knowledge due to its notation and the inherent complexity of designing ontologies. ASLAN "shields" the designer from Ontolingua. ASLAN accomplishes this "shielding" by allowing the designer to specify knowledge in a meaningful, intuitive way. This (what?) is then transformed into an Ontolingua ontology. Knowledge acquired via ASLAN can then be encoded into a KQML (Mayfield, Labrou, and Finin 1996) application programming interface using KQML's performatives and Ontolingua's content format. This knowledge can then be shared with the agent research community at large and used within an interface agent-based system. This approach is similar to AgentBuilder's (Reticular Systems 1998).

6.3 Addressing Agent Development Environment Deficiencies

Current agent development environments typically focus on collaborative, autonomous multi-agent system specification and development (more the former than the latter). However, they tend to ignore the fact that many agents, in particular interface agents, need the ability to adapt to the changing needs of the user and environment. Furthermore, as mentioned earlier, it is desirable to develop generic intelligent user interface agents usable in many different domains. Generic interface agent development requires a robustness not found in other development environments.

The Intelligent Agent Development Environment Architecture (IaDEA) depicted in Figure 6 explicitly addresses the deficiencies of existing agent development environments. The human interface developer's specifications, to include the interface agent user model, user interface standards, and the target system application programmers interface are specified using ASLAN. ASLAN is responsible for trans-

105

forming the Ontolingua-based specification into interface agent source code and the user model. These two components are used by the CIA architecture. Additionally, the CIA architecture uses the requirements and metrics defined in Section 5.1. The resulting CIA architecture can be used as a prototype that can be validated against the human interface developer's specifications. This feedback process allows the developer to refine the specification. The "final" customized intelligent user interface is an integration between the user, target system, and CIA architecture as depicted in Figure 3.
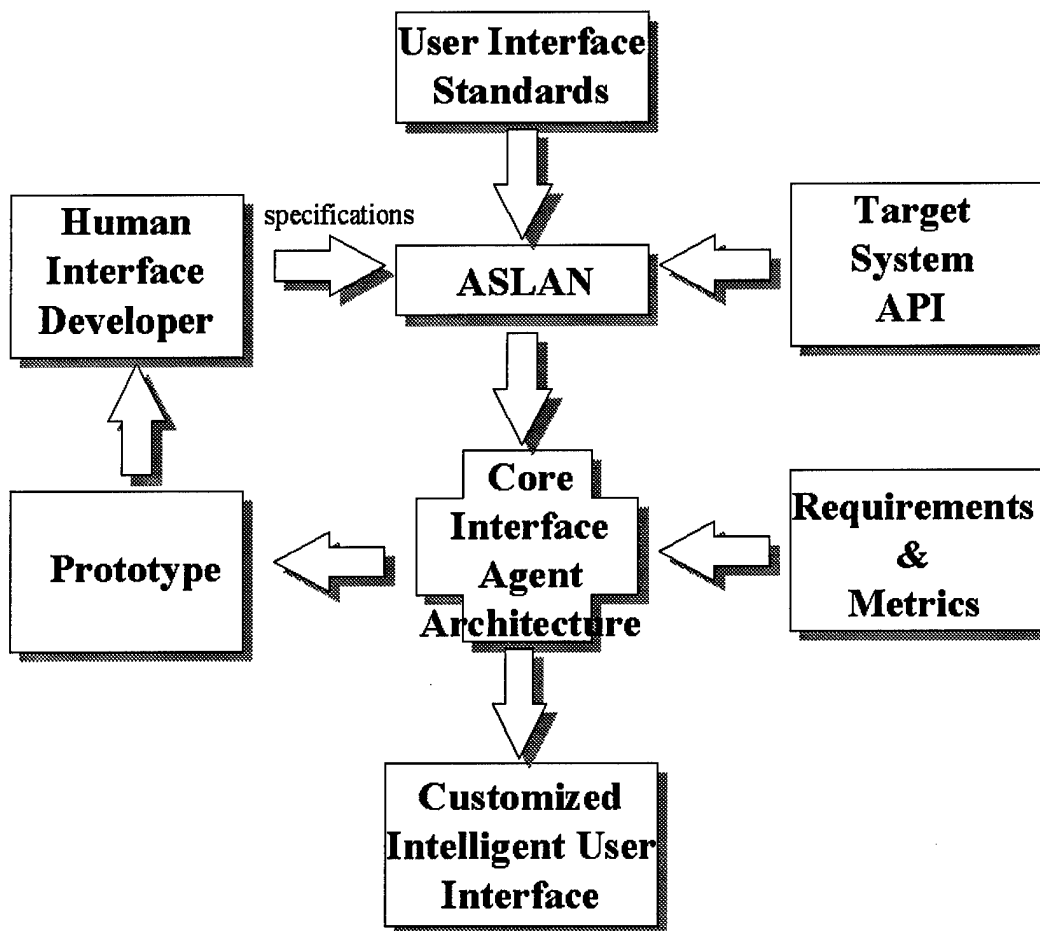

Figure 6   Interface Agent Development Environment Architecture (IaDEA): High-level process flow for construction of customized intelligent user interfaces.

106

This section proceeds to describe how IaDEA can be used to address the above three deficiencies (i.e., environmental specification, adaptivity mechanisms, and agent knowledge representation and reasoning mechanisms) of existing agent development environments.

IaDEA Environment Specification. To specify a target system's environment, IaDEA must specify what events are expected from the user interface, i.e., the stimuli the user interface can provide to the agent. For example, the interface agent needs to be able to determine when a button is pressed, a menu item is selected, the user exits the system, etc. Additionally, the agent must know how it can affect the environment where it is situated, i.e., the affectors. These affectors can either be defined by the target system's API or assumed that the designer of the agent will have direct access to the target system's source code. ASLAN can be used to elicit an explicit enumeration of the "call-backs" the agent can use as input from and output to the application. This enumeration is sufficient to capture the application specific stimuli and affectors.

ASLAN supports the incorporation of user interface standards and business practices and work flows into the specification. An approach similar to Stary (1997) can be used. Stary's TADEUS (Task Analysis, Design, End-User Systems) approach, deals with designing intelligence interfaces by addressing what a user needs for task accomplishment. Work flows are migrated into the user-interface design representations. Business goals and rules and their relationships to tasks and people involved can be integrated into the user interface design. Meaningful sequential activities for task accomplishment are determined. Profiles of various users of the system, and their functional roles to the task and work flows can be generated. Flow and control of data can be made transparent and interface designers can provide notation that allows static and dynamic specification and adaptation of the work flow models. The TADEUS approach allows consistent and context-sensitive user interface development to be supported in the software development process and eases the use of

artifacts of previous development. The TADEUS approach does not consider the use of an interface agent to accomplish tasks on the user's behalf. The SIRDS approach explicitly considers the partitioning of work tasks between the interface agent and the user. Therefore, the TADEUS approach can be enhanced to include specification of those tasks the interface agent should handle.

Interface agent development must explicitly deal with one component other agent systems possibly ignore — the human agent. Human users have beliefs, desires, intentions, abilities, preferences, emotions, plans, and goals. To offer beneficial assistance to the user, the interface agent user model must be able to capture these characteristics. Furthermore, the environment the interface agent and human user are situated in has a direct impact on the user's behaviors as well as the environment being affected by these two agents. This interaction between environment, user, and interface agent must be captured.

The ability to accurately capture user characteristics is difficult to achieve. For example, there is typically not a specific user action within the environment that determines a certain user characteristic (e.g., intentions). However, several techniques from the user modeling research field are applicable, e.g., user profiles and stereotypes. For example, Van Veldhuizen et al. (1998) use a "skills vector" to define an agent's ability to perform certain tasks, measured by standardized scores. Human factors can be incorporated into user models (Mulgund and Zacharias 1996; Schäfer and Weyrath 1997; Stefanuk 1997; Gavrilova and Voinov 1997).

As a first step towards modeling user characteristics, ASLAN can function as a knowledge acquisition front end to Ontoligua to specify both a user profile for a specific user and user stereotypes for classes of users. Any of the aforementioned techniques for specifying user profiles and stereotypes is suitable for use within IaDEA. With regards to specifying user intent, since a causal relationship between environmental stimuli, human factors, users' goals, and the actions users perform, ASLAN can use a directed acyclic graph to show this causality. Furthermore, since ascribing

user intent is inherently uncertain, probabilities can be assigned to the arcs in the directed graph.

IaDEA Adaptivity Mechanisms. As previously mentioned, a key distinguishing factor between software and agent-based engineering is that agent-based engineering deals with dynamic environments. However, as is evident from the existing agent development environments, adaptivity to dynamic environments is not often explicitly considered. IaDEA brings the requirement of adaptivity to the forefront of agent specification and design, guiding the designer to areas in the design where adaptivity is warranted. ASLAN is responsible for specifying those components used in the CIA architecture's correction model. In particular, ASLAN allows the developer to specify the metrics and metric weights used in Equation (16). ASLAN also allows the developer to specify the requirements utility function value threshold, as well as the bid deadline. The collection of correction adaptation agents can be presented to the developer. The developer can determine which correction adaptation agents to include in the CIA architecture. The reuse of correction adaptation agents for multiple domains supports rapid prototyping.

IaDEA Agent Knowledge Base and Reasoning Mechanisms. ASLAN uses Ontolingua. Ontolingua is not meant as a representation language for reasoning. For efficiency's sake, a different knowledge representation should be used for reasoning. The ontology defines an adaptive intelligent agent, its environment (a user interface), and the user's goals and associated actions to achieve those goals within that environment. Since the user model's knowledge representation is probabilistic, a representation that can handle probabilistic reasoning is needed. Additionally, as mentioned previously, a causal relationship exists among environmental stimuli, users' goals, and actions. This causality is used to construct Bayesian networks (Pearl 1988) based on the observable events and goals within the environment (Jameson 1996) as described in Chapter III.

6.4 ASLAN - CIA Architecture Interaction

With input/output user interface standards and requirements — such as DII-COE compliance and those imposed by the particular software system we are constructing the interface for — fully specified in the ASLAN environment, surface issues are separated from the core issue of how such an interface agent should perform. Each component of the CIA architecture was discussed in detail in Chapter IV. Recall Figure 3 gives an overview of CIA architecture including user intent prediction and continual adaptivity. The task of ascribing user intent is delegated to the interface agent component of the architecture, while continual adaptation of the interface agent's user model is a task shared by the interface and and a collection of correction adaptation agents.

Furthermore, Figure 4 shows the architecture of the interface agent and the correction adaptation agents. The content of the KQML messages is specified via ASLAN. Each target system observation (environmental stimuli, user action, and human factor) can be communicated to the agents via the KQML message passing API. Every observation is stored by the agents' evaluator in a history stack (i.e., most recent observation is on the top of the stack). These observations can be used by the agents as evidence into the user model, represented as a Bayesian network. The interface agent offers assistance via suggestions to the target system (user) by calculating the expected utility of offering assistance for a goal, $EU(\alpha_G)$. $EU(\alpha_G)$ is calculated by performing Bayesian network belief updating on the goal random variable and the utility of suggesting the actions used to achieve the goal, $U(G, a, \Delta)$.

6.5 Conclusion

The need for development methodologies for agents is readily apparent. The approach outlined in this chapter is to use the Interface agent Development Environment Architecture (IaDEA), with ASLAN explicitly specifying the environment, to include the target system API, applicable user interface standards, user model,

and human factor concerns and the CIA architecture continually adapting the user model to the needs of the user. I do not mean to presuppose that this approach is the Holy Grail of interface agent development. Implemention of the approach presented here is sure to uncover unforeseen problems and pitfalls. However, the approach addresses deficiencies of current agent development environments — environment specification, adaptivity, and agent knowledge base and reasoning mechanisms. It is desired to easily and directly incorporate business practices and user interface standards into ASLAN to produce compliant, intelligent interface agents. In other words, designers can treat user interface standards as simply an input database to define ASLAN. This encapsulation of the input and outputs of the CIA architecture isolates the architecture from these concerns.

VII. Experiments

In this chapter, three disparate domains are presented — a virtual space plane, a probabilistic expert system shell, and a natural language query information management system — that were used as test beds for the interface agent research presented in this dissertation. These three domains were chosen for a number of reasons. First, all three projects are in-house research projects and the target system source code was easily accessible. The virtual space plane and probabilistic expert system shell domains provide environments with many interaction modes and tools users can utilize to perform thier tasks. Furthermore, the two domains are used by a wide diversity of users. These users can be stereotyped into one of several groups (e.g., pilot or navigator for the virtual space plane and knowledge engineer or user for the probabilistic expert system shell). Yet, individual users have preferences in how they interact with these domains. Therefore, these two domains provide a way to test the CIA architecture's ability to offer timely, beneficial assistance to users and to adapt that assistance to the different needs of users. The natural language query information management system tests the CIA architecture's ability to dynamically construct and adapt a user model over time as the environment (i.e., the information source being managed) changes over time.

7.1 Preliminary Experiment

The first experiment performed was designed to test the approach for user intent ascription presented in Chapter III. As discussed in Section 2.5, a knowledge engineer must determine how to construct the Bayesian network to best represent causality within the domain. The approach taken in this research is to observe atomic user actions and pre-conditions within an environment to determine what goals a user is pursuing so as to help the user perform the other associated actions to achieve

that goal. An experiment designed to validate the approach must be centered on the user models created by following this approach.

Two simple user models were constructed as shown in Figure 7. All random variables in the networks are Boolean (i.e., true and false states) and the complete definition of the network is given in Appendix E. The construction of the two user models is similar to those presented by Jameson (1996, pp. 8–9). For the experiment, action $A2$ was set as evidence and belief updating was performed on all remaining nodes. Based on the discussion given by Jameson as well as general discussion on the effects of evidence on *posterior* probabilities (i.e., the conditional probability of a random variable after belief updating is performed) in Pearl (1988), results can be predicted. In Figure 7, the upper Bayesian network can be predicted as follows: observation that a user is pursuing a goal increases the likelihood (i.e., probability) that the user will perform an action to achieve that goal. The semantics of the construction of the upper network is that the interface agent is concerned with the conditional probabilities of the goal nodes given pre-condition or action evidence. The lower Bayesian network represents the approach discussed in Chapter IV, namely causality proceeds from pre-conditions ($P1$ and $P2$ in Figure 7) to goals ($G1$ and $G2$) to actions ($A1$, $A2$, and $A3$) to achieve those goals. Since the interface agent performs keyhole plan recognition, it never observes a user performing a goal explicitly[1]; the agent infers (via the decision-theoretic approach) a user is pursuing a goal given the observational evidence. Therefore, intuitively, the lower Bayesian network user model better represents the approach in Chapter III. The experiment was designed to verify this intuition quantitatively. From the discussion given by Jameson, observed evidence "propogates" only downward for networks constructed like the upper network in Figure 7. For networks constructed like the lower Bayesian network, evidence propagates upward and then downward.

---

[1]For a way to relax this restriction, see Chapter VIII.
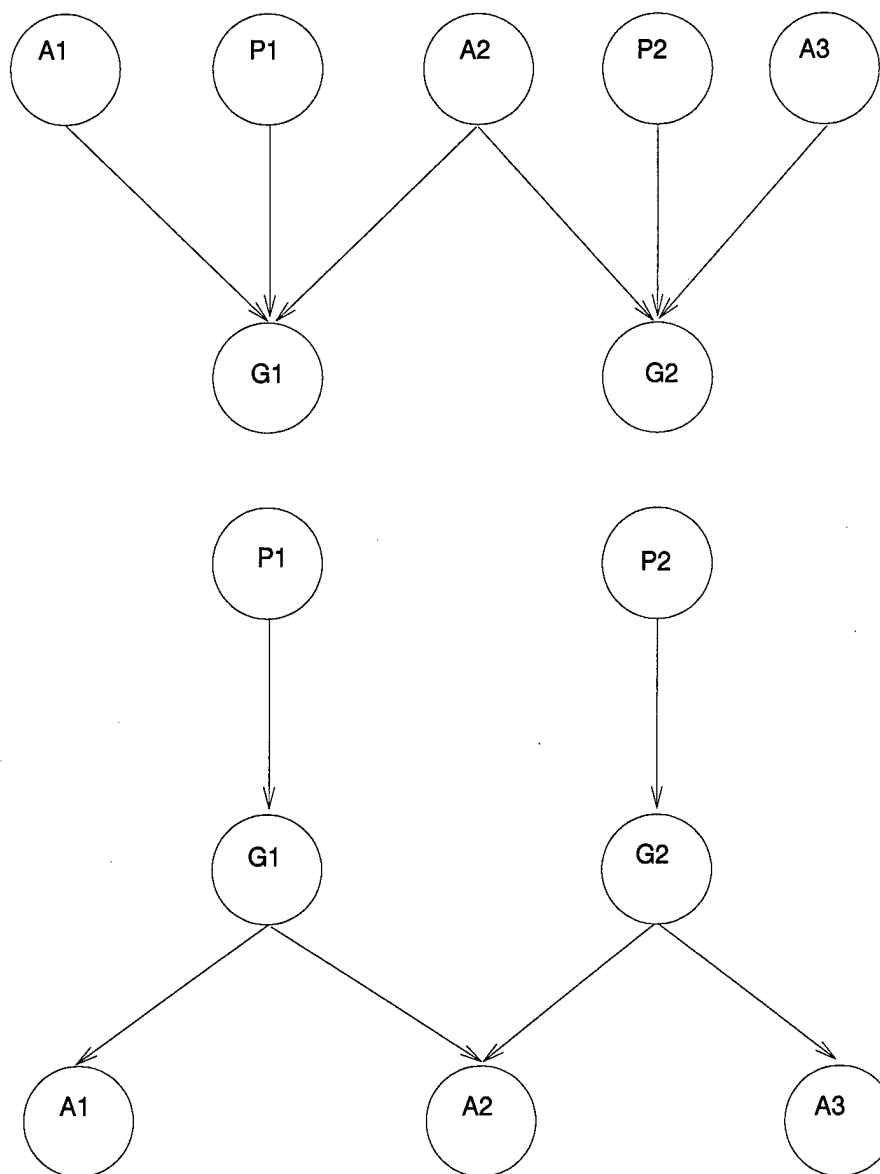
Figure 7   Two Simple User Models to Determine Causality Direction.

Table 1 and Table 2 show the *prior* and *posterior* conditional probabilities for all nodes in the two networks. The probabilities given are for $\Pr(RV = true)$. To obtain $\Pr(RV = false)$, the formula $\Pr(RV = false) = 1 - \Pr(RV = true)$ is used. As predicted, for the upper network in Figure 7 the probability of goals $G1$ and $G2$ increased but the other nodes' probabilities did not increase. However, for the lower network, observing $A2$ not only increased the probabilities of goals $G1$ and $G2$, but

*A*1, *A*3, *P*1, and *P*2 as well. The results were therefore the same as the expected results and the discussion presented by Jameson. These results justified the general construction of the user model (from pre-conditions, to goals, to actions).

Table 1: Prior and Posterior Random Variable (RV) Probabilities for Upper Bayesian Network Given Evidence *A2* (Pr(RV = true) only).

RV	Prior	Posterior
A1	0.70	0.70
A2	0.35	1.00
A3	0.75	0.75
P1	0.55	0.55
P2	0.45	0.45
G1	0.40	0.67
G2	0.49	0.55

Table 2: Prior and Posterior Random Variable (RV) Probabilities for Lower Bayesian Network Given Evidence *A2* (Pr(RV = true) only).

RV	Prior	Posterior
A1	0.54	0.83
A2	0.51	1.00
A3	0.50	0.64
P1	0.55	0.84

Table 2: Prior and Posterior Random Variable (RV)
Probabilities for Lower Bayesian Network Given Evi-
dence *A2* (Pr(RV = true) only).

RV	Prior	Posterior
P2	0.25	0.60
G1	0.55	0.86
G2	0.45	0.61

7.2 Virtual SpacePlane

Research in the field of intelligent interface agents for virtual environments
was first demonstrated by the CIA architecture's integration into a virtual space-
plane environment (Stytz and Banks 1997; Brown, Santos Jr., Banks, and Stytz
1998b). The Virtual SpacePlane (VSP) is a prototype of the Manned SpacePlane
(MSP), a spacecraft capable of supporting the United States Air Force's mission of
providing worldwide deployment of space assets with minimal preflight and in-orbit
support from a mission control center. The goals of the VSP project are to uncover,
develop and validate the MSP's user interface requirements, develop a prototype vir-
tual spaceplane to demonstrate MSP missions, and to conduct preliminary training
experiments. The VSP environment is an accurate, high fidelity presentation of the
the Earth's surface as seen from orbit, and the contents of the space environment.
The architectural design of the VSP allows rapid prototyping of the cockpit's user
interface and flight dynamics.

The CIA architecture was integrated into the VSP to support VSP user as-
sistance such as real-time information visualization and automation of the landing
sequence. Figure 8 shows the preliminary integration of the interface agent within
the VSP environment. Here, the interface agent has suggested the user land at Ed-

wards Air Force Base based on a number of observable environmental stimuli. If the user chooses to allow the agent to achieve this goal (by clicking on the "ok" button), the agent will perform the necessary actions to land the spaceplane. As a result of lack of resources, the user model, shown in Figure 9, was simple and covered a very small portion of the entire domain.
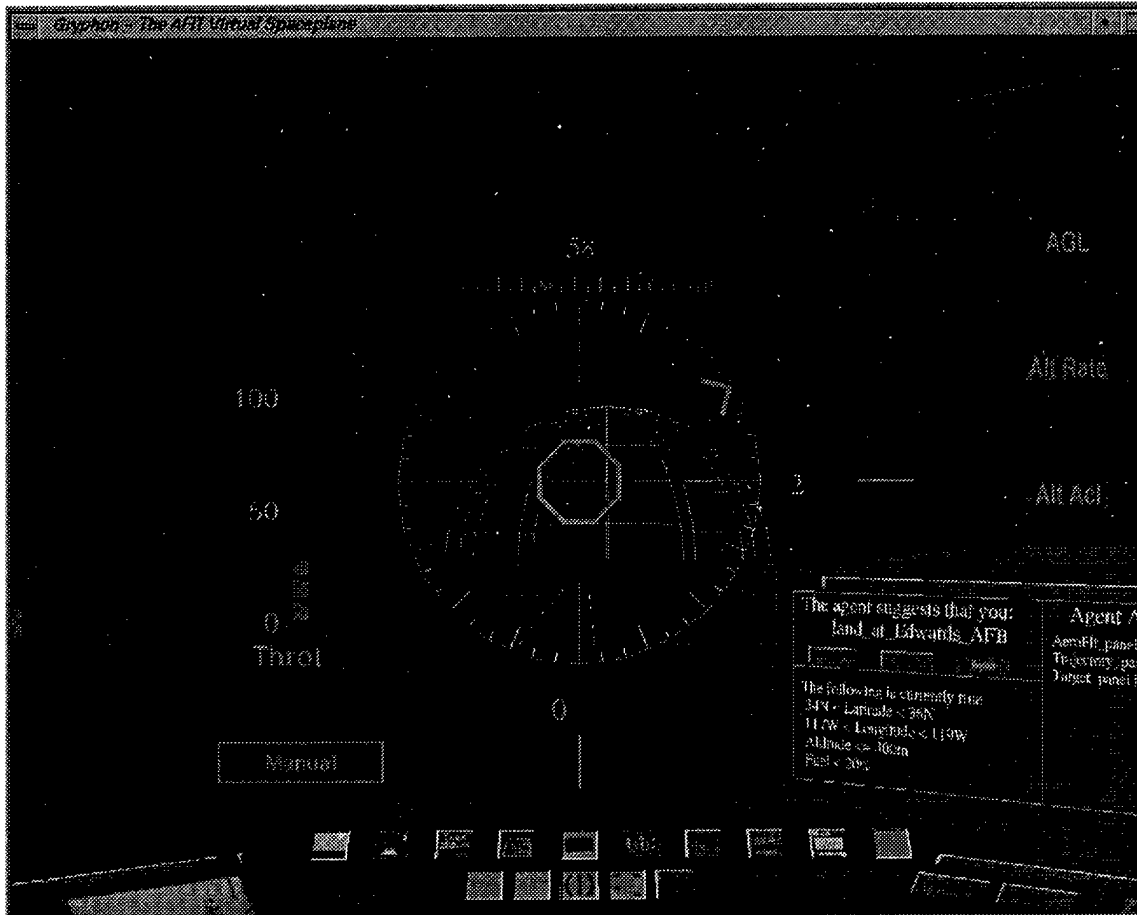

Figure 8   The Virtual SpacePlane with an Interface Agent.

The VSP was the first domain to benefit from the integration of the CIA architecture. As a result, a number of user model design issues were uncovered. These design considerations, discussed next, were taken into account in the other two domains.

The goal and action random variables (RVs) of the Bayesian network user model are Boolean. That is, the two states for the goal and action RVs are *true* and *false*. These RVs are set as evidence when they are observed. As a result, the names must be chosen carefully to convey the proper semantic meaning of the observation. For example, in the virtual space plane, "panels" can be open (i.e., viewed) or closed. The action RVs associated, for instance, with the agent panel is named *"Open_Agent_Panel"* with a true state indicating the panel is open. Therefore, to close the agent panel, the interface agent must suggest the value of *"Open_Agent_Panel"* be set to false. If the Bayesian network was designed to allow the action RVs to have non-Boolean states (for example, allowing for one action RV for all panels with states for each panel), the interface agent would not be able to suggest opening multiple panels since states of a single RV are mutually exclusive.

There are times, however, when it might make more sense to combine several action RVs into one RV with non-Boolean states. One example is the communication modes in PESKI (described in Section 7.3). A tool in PESKI can operate in only one communication mode (e.g., text, natural language, or graphical). Therefore, a knowledge engineer could design one RV, "Communication Mode" with states "Text," "Natural Language," and "Graphical." However, early in the design of the CIA architecture it was decided that goal and action RVs would be Boolean. The extra logic needed to allow for non-Boolean RVs was not warranted. Furthermore, explanation generation of agent assistance is easier when goals and actions are Boolean. To handle the "Communication Mode" situation (and obviously others), an OR sub-goal[2] is created with one action RV for each of the mutually exclusive actions. So, for the communication mode example, the solution is to create an OR sub-goal named "Communication Mode" with three action RVs named "Text," "Natural Language," and "Graphical." The interface agent chooses the one action with the maximum expected utility.

---

[2]The OR sub-goal functions as a logical XOR.

Another design technique was the segregation of human factor observables to the utility model. As Figure 9 shows, cognitive load was a pre-condition to the goal "reduce cognitive load." Although the reduction of cognitive load is a goal (in this case, a sub-goal) during landing, cognitive load, as well as other human factors, is better captured by the user's utility model. The reason to relegate the human factors to the the user's utility model is twofold. First, human factors (e.g., skill, expertise, fatigue, workload, temporal and spatial memory capacity, preferences, biases, prejudices, etc.) all affect the user's decision to pursue goals and perform actions in pursuit of goals. A user's utility model captures the influence these factors have on the decision making process. Second, human factors do not fit well with the Bayesian network user model construction approach based on the "pre-conditions to goal to actions" philosophy. Human factors could be considered pre-conditions (certainly they are not goals or actions). Recalling the discusion in Chapter III, pre-conditions are directly observable events and/or stimuli in the environment. These pre-conditions cause a user to pursue a goal and/or affect the goal a user will pursue. Human factors are not events or stimuli. Human factors are not typically directly observable but they are measurable, either *a priori*, such as skill or expertise, or dynamically as the user interacts with the environment, e.g., workload. They are influencing factors on the user's decision making process and thus, as the first reason argued, belong in the user's utility model.

7.3 Probabilities, Expert Systems, Knowledge, and Inference

Most everyday decisions involve some level of uncertainty. Expert systems, also known as knowledge-based systems, attempt to capture an expert's knowledge for use by non-experts. Among the advantages in using expert systems are wide distribution, accessibility, and preservation of scarce expertise; ease of modification, consistency and explanation of the answers (Gonzalez and Dankel 1993). One of the greatest disadvantages of expert systems is their construction. To aid experts in the
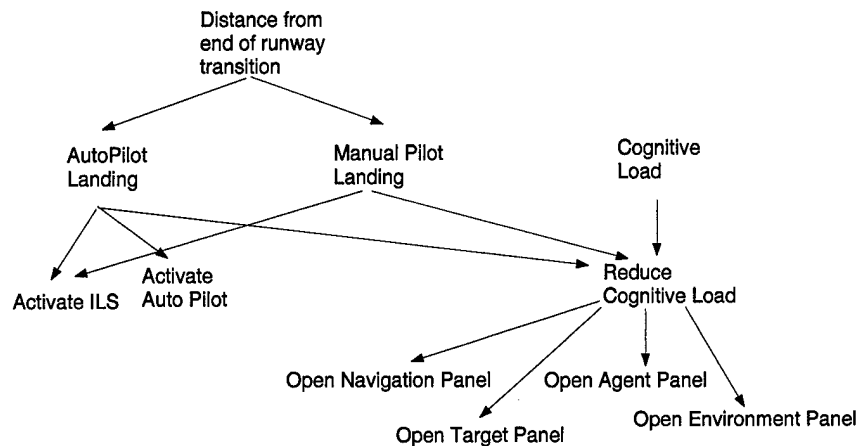
119

Figure 9   The user model for the Virtual SpacePlane.

arduous task of designing expert systems, a number of expert systems shells exist. Most of these shells allow the expert system designers to capture an expert's knowledge, verify and validate that knowledge, and query this knowledge, i.e., perform inference. The tools available within a system vary between each shell. Some provide a graphical means of acquiring knowledge from users. Most incorporate some form of verification and validation of the knowledge. However, none of these systems provide an integrated suite of tools for acquiring knowledge, testing that knowledge via verification and validation, and inference. Furthermore, these systems typically require complete information before they are of any use.

The usefulness of the CIA architecure for providing assistance to users of expert system shells is demonstrated by integration of the CIA Architecture into an expert system shell called PESKI (Harrington, Banks, and Santos Jr. 1996a; Harrington, Banks, and Santos Jr. 1996b; Brown, Santos Jr., and Banks 1999). PESKI (Probabilities, Expert Systems, Knowledge, and Inference) is an integrated probabilistic knowledge-based expert system shell (Brown, Santos Jr., and Banks 1998a). PESKI provides users with knowledge acquisition (Santos Jr., Banks, and Banks 1997), verification and validation (Bawcom 1997; Santos Jr., Gleason, and Banks 1997), data mining (Stein III, Banks, Santos Jr., and Talbert 1997), and inference
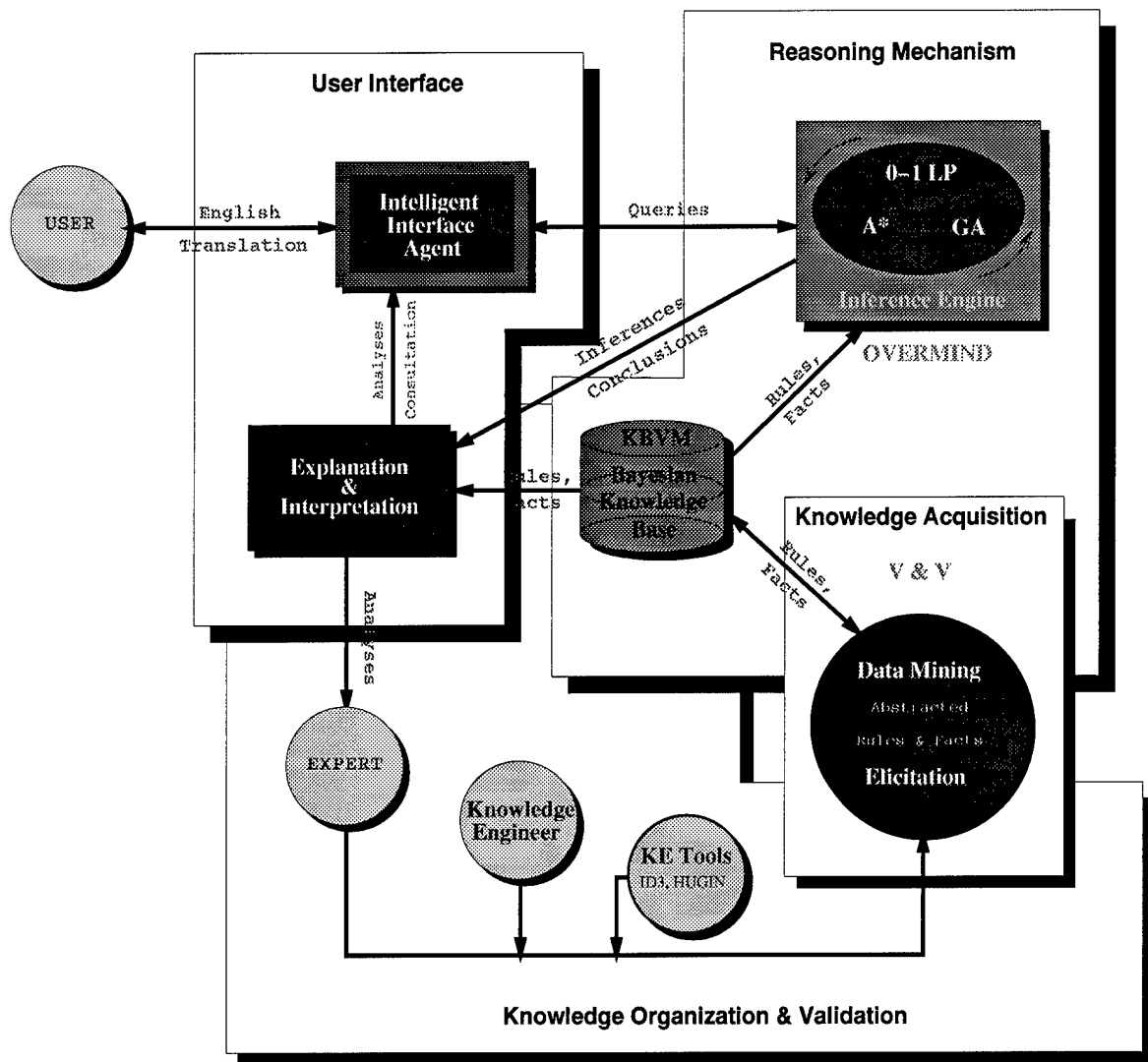
Figure 10   The PESKI Architecture.

engine tools (Shimony, Domshlak, and Santos 1997), each capable of operating in various communication modes. Figure 10 shows the PESKI architecture.

The architecture consists of four major components as shown in Figure 10:

1. **Intelligent Interface Agent** — Translates English questions into inference queries and translates the analyses/inference results back into English; provides for the communication exchange between the user and the system; provides intelligent assistance to the user.

121

2. **Inference Engine** — Contains the intelligent control strategies for controlling the selection and application of various inference engine algorithms (e.g. A*, 0–1 integer linear programming (ILP), genetic algorithms (GAs)); obtains conclusions to user queries based on knowledge and facts in the knowledge base (Williams 1997).

3. **Explanation and Interpretation** — Tracks the reasoning paths the inference engine used in reaching its conclusions; allows the user to query the system about how and why an answer was derived.

4. **Knowledge Acquisition and Maintenance** — Provides the facility for automatically incorporating new or updated expert knowledge into the knowledge base.

Architecturally speaking, PESKI is divided into three subsystems. The four components perform multiple functions and each PESKI subsystem combines different components together for that subsystem. The subsystems as they occur in PESKI are as follows:

1. **User Interface** — Composed of the Intelligent Interface Agent and the Explanation and Interpretation components, as well as direct manipulation interface components.

2. **Knowledge Organization and Validation** — Consists of the Explanation and Interpretation component along with the human expert, optional knowledge engineer and knowledge engineering tools. Organization is accomplished by communicating with the Knowledge Acquisition and Maintenance component, ensuring compliance with the BKB consistency constraints. Validation is similarly accomplished except that PESKI also has feedback from the Reasoning Mechanism through Explanation and Interpretation for debugging purposes.

3. **Reasoning Mechanism** — Consists of the Inference Engine and the Knowledge Acquisition and Maintenance components. The premise behind incorporating the Knowledge Acquisition and Maintenance component is that some form of reasoning and possibly learning must take place in order for any new knowledge to be merged into the existing knowledge base. Problems such as consistency must be dealt with. Furthermore, we can also achieve a useful degree of information hiding. Under this arrangement, it is not necessary for any other subsystem outside of the Reasoning Mechanism to concern itself with the particular choice of knowledge representation. One exception is the Explanation and Interpretation component; however, it only needs to read and interpret knowledge-base information.

7.3.1 PESKI's Integrated Tool Suite. Figure 11 shows a user utilizing the PESKI verification and validation, inference engine tools, and data mining. The tools integrated into the PESKI architecture are described next.

- **Knowledge Acquisition** — PESKI uses the MACK tool for knowledge acquisition (Santos Jr., Banks, and Banks 1997). MACK contains routines designed to automatically and incrementally confirm consistency of the knowledge elicited from the expert and provides assistance via knowledge base status messages. Regular incremental checks preserve both probabilistic validity and logical consistency by flagging the inconsistent data points to the expert as they are entered.

- **Verification and Validation** — PESKI verification and validation is performed using two tools: BVAL (Santos Jr., Gleason, and Banks 1997) and a graphical incompleteness tool (Bawcom 1997). BVAL validates a knowledge base against its requirements using a test case-based approach. A test case is a set of evidence and expected answers. A knowledge engineer submits a test case to the BVAL tool and BVAL determines if the inference engine can obtain

123

the expected answer given the evidence. Under certain conditions, the knowledge base can be corrected via reinforcement learning of the probabilities. If these conditions are not met and incompleteness exists (typically the result of a missing causal relationship between two random variables), the graphical incompleteness tool may be used to visualize the knowledge base incompleteness and correct it. The tool uses data visualization of the BKB and data mining to assist the user in eliciting the needed knowledge.

- **Inference Engine** — The PESKI inference engine uses a performance metric-based approach to intelligently control a number of possible anytime and anywhere inferencing algorithms. Results are returned to the user via the Explanation and Interpretation subsystem of PESKI as they become available.

- **Data Mining** — PESKI uses a goal-directed methodology for data mining for association rules and incorporation of these rules into the knowledge base (Stein III, Banks, Santos Jr., and Talbert 1997). Data mining within PESKI can either be a knowledge acquisition, or verification and validation process. In the latter case, an expert attempts to correct problems discovered as a result of performing verification and validation. In the former, using empirical and/or legacy data, an expert is able to mine for specific rules relating two or more database attributes (i.e., random variables in the BKB). Additionally, the data mining tool can be used to find new states of a random variable and to elicit the probabilities of a single state.

Each tool in PESKI displays the current status of the BKB, alerting the user to any problems with the knowledge base. PESKI supports incremental knowledge elicitation in a number of ways (Santos Jr., Banks, and Banks 1997). During knowledge acquisition, the user is alerted to any inconsistencies in the BKB knowledge representation. For example, if the user attempts to add a rule that creates a cycle in the knowledge base, PESKI displays an error message to the user.

124
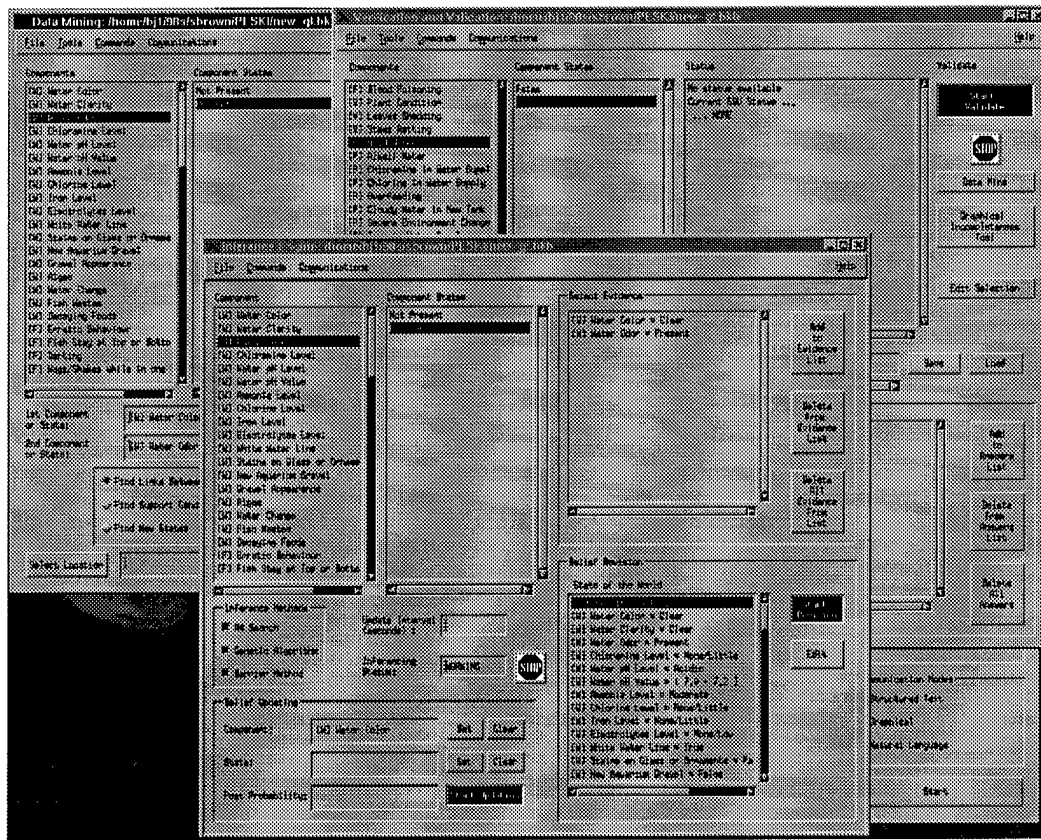
Figure 11   PESKI Tool. This is an example of a user utilizing the Verification and Validation, Inference Engine, and Data Mining tools.

7.3.2   PESKI's Intelligent Assistance Experiment.   Determining which tools to use given a particular situation in PESKI is difficult for most users. The use of a particular tool is dependent on a number of variables including the context (e.g., a BKB constraint violation exists) and user preferences for the tools and various communication modes. Determining the correct tool to use at the correct time can be a daunting task.

The main purpose for using PESKI as an application domain was to test the CIA architecture's ability to offer timely, beneficial assistance that was adaptable to the needs of different users. To be precise, an experiment was designed to test the dissertation hypothesis that the CIA architecture is an effective and efficient

decision-theoretic architecture for user intent ascription and that the architecture improves the interface agent's utility for providing assistance to the user.

The experiment consisted of two different users: an expert user and a novice user. The users' user model profiles are shown in Table 3. The users' Bayesian network user models were structurally the same. The conditional probabilities for the novice's user model "favored" using the graphical communication mode, whereas the expert's perferred the structure text communication mode. Furthermore, the following two equations specify the two users' requirements utility functions:

$$
\begin{aligned}
U_{requirements}^{novice} = \ & 0.1428reactive + 0.1428precision \\
& +0.1428assistance\ capability + 0.1428external\ autonomy \\
& +0.1428collaboration + 0.1428perceptive \\
& +0.1428misconception
\end{aligned}
\tag{24}
$$

and

$$
\begin{aligned}
U_{requirements}^{expert} = \ & 0.0856reactive + 0.2precision \\
& +0.1428assistance\ capability + 0.1428external\ autonomy \\
& +0.1428collaboration + 0.0856perceptive \\
& +0.2misconception.
\end{aligned}
\tag{25}
$$

See Section 5.1 for the definition of the metrics used in the two equations.

Previous research done with PESKI identified four groups of stereotypical users (Harrington, Banks, and Santos Jr. 1996a): an application user (i.e., novice), an application expert, a computer scientist, and a knowledge engineer. The user profiles in Table 3 are stereotypical of the novice and expert users. These two users were given the task of constructing a small Bayesian knowledge base. The expected result of the experiment was that the assistance offered to the two users would be

different, based on their user models and usage patterns (i.e., the actions they performed). Since the novice user has lower assistance thresholds and requirements utility function value threshold, it was expected that the interface agent would offer more assistance to the user. The expert user's high assistance thresholds indicate this user prefers little assistance unless the assistance is extremely beneficial (i.e., of high utility). The requirements utility functions for the two user's are different as well. The novice user places equal weights on all requirement metrics, whereas the expert user places greater importance (i.e., weight) on the agent's ability to make correct suggestions (i.e., the predictive and misconception metrics) while placing less importance on how quickly those suggestions can be made (i.e., the reactive and assistance capability metrics).

Table 3: PESKI user model parameters for an expert and novice user.

Parameter	Expert	Novice
Autonomous Threshold	0.85	0.75
Collaborative Threshold	0.50	0.25
$U_{requirements}$ Threshold	0.50	0.40
Cut-off time	15 seconds	15 seconds
Bid deadline time	120 seconds	120 seconds
Expertise	0.85	0.25
Operative memory	0.75	0.25
Spatial memory	0.65	0.50
Temporal memory	0.75	0.45
Response time	0.75	0.45

7.3.3 PESKI Results and Analysis.   The assistance offered by the CIA architecture while the two users were performing the task of constructing the Bayesian knowledge base was beneficial and was notably different for the two users. As expected, more assistance was offered to the novice user as the result of the novice user's assistance thresholds. Conversely, less assistance was offered to the expert user. In general, the value of the novice user's requirements utility function was greater than the expert's value.

The reason the expert user received less assistance was the result of the combination of expert's higher assistance thresholds and requirements utility function. As discussed in Section 5.1, the precision and assistance capability metrics depend on the ability of the interface agent to determine a correct suggestion was made. As a result of the expert's high assistance thresholds, the situation existed where little assistance was offered. Therefore, the interface agent was unable to determine if the assistance being offered was correct or incorrect. This resulted in the requirements utility function value being lower than the expert's requirements utility function value threshold which prompted a bidding process to correct the user model. The CIA architecture does not offer assistance to the user if the requirements utility function value is not above the requirements utility function value threshold. This analysis also explains why the value of the novice user's requirements utility function was greater than the expert's value.

7.4 Clavin

The Clavin System is the next step towards realizing the SIRDS vision. Clavin is an intelligent natural language query information management system. The Clavin System architecture is shown in Figure 12. The system consists of four main components. The human language interface (HLI) component is composed of a commercial off-the-shelf voice recognition system[3] and a natural language (English) parser. The

---

[3]IBM's Via Voice was used for this system.

voice recognition system is responsible for transforming a spoken utterance by the user into a natural language sentence. The natural language sentence is then parsed into an S-expression predicate logic representation of the user queries by the natural language parser (NLP) component. The well-formed formula (wff) query is then passed to the interface agent.

Clavin responds to users' natural language inquiries by forming intelligent queries to a database of dynamic, heterogeneous information sources. Clavin maintains a dynamic user model of the relevant concepts in the user inquiries as they relate to the information sources. The user model allows Clavin to determine the relevancy of the various concepts in the domain, modifying user inquiries. Clavin autonomously reacts to changes in the information sources, alerting the user to relevant changes in the state of the world. Furthermore, Clavin is capable of proactively retrieving relevant information for the user based on the current inquiries and the user's previous history of inquiries.

An example of an uttered query, its wff representation, and the resulting representation in the user model is shown in Figure 13. Once the interface agent processes the query (see the discussion below), the query is passed to the retrieval engine. The engine parses the query wff and transforms it into a database query. Heterogeneous, possibly dynamic information sources are then queried by the retrieval engine. These information sources are represented to the user as one homogeneous information source. Results are then passed back to the interface agent for data visualization of the resulting query. If no results are returned, the interface agent can request the HLI to produce another possible parse (i.e., re-tag the parts of speech) for the uttered query.

The CIA architecture is responsible for two key functions within the Clavin system: information filtering and proactive querying. The first occurs as a result of adding context to the spoken queries. The second occurs as a result of combining various relevant pieces of previous queries. For example, if the user asks "What
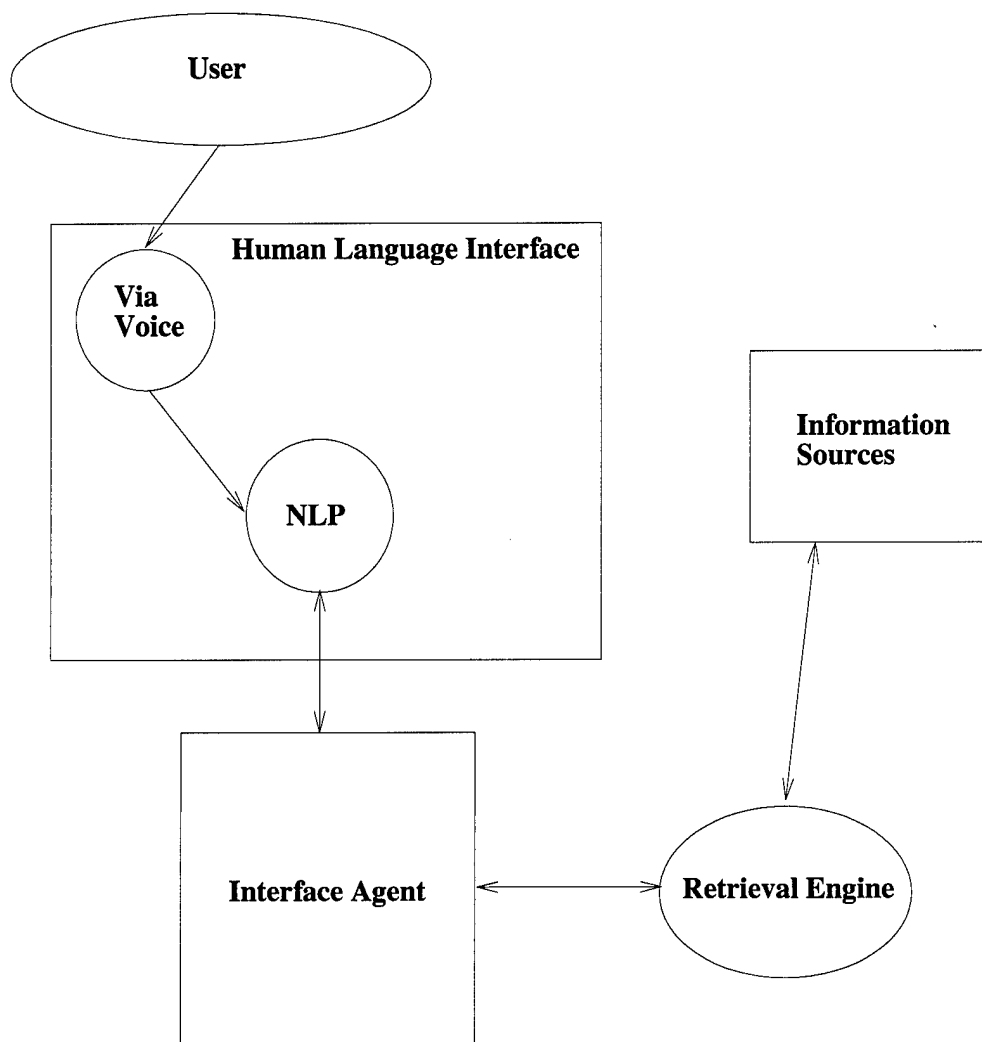
Figure 12   The Clavin System Architecture.
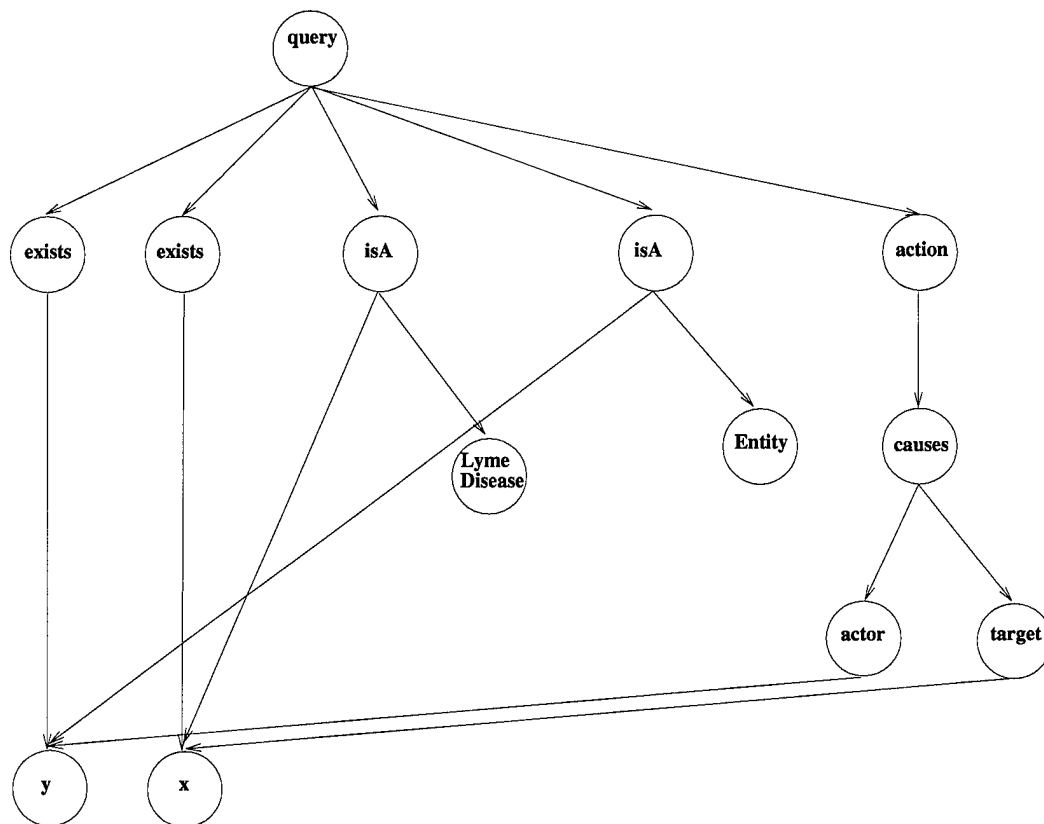
Figure 13   A User Model Representation of the Spoken Query "What causes Lyme
Disease?". The utterance is transformed to the wff (exists x)(exists
y)(isA x Lyme Disease)(isA y Entity)((action causes)(actor y) (target
x)).

causes Lyme disease?" the system returns information about deer tick bites; then

the user's next query asks "What treats Lyme disease?", to which the system replies

with information concerning a new Lyme disease vaccine. At this point, the user

model contains information about the concepts "Lyme disease," "tick bite," "Lyme

disease vaccine," "causes," and "treats." If the user then makes a inquiry about

"what causes cancer?" the interface agent component can use the CIA architecture

to not only retrieve information about the causation of cancer, but also proactively

query the database about possible treatments for cancer. The proactive construction

of queries is discussed in the next section.

Within the Clavin architecture, information sources can be of a diverse variety. These sources may include, but are not limited to, natural language text, Hypertext Markup Language (HTML), World Wide Web news feeds, Usenet newsgroups, and "standard" databases. To adequately test the Clavin architecture, it was desired to have a large information source as a rich source of real world knowledge. The Unified Medical Language System (UMLS) provided just such an information source.

The UMLS approach is to provide a set of widely available medical-related knowledge sources usable by a variety of application programs. The UMLS consists of three main components: a Metathesaurus, Semantic Network, and SPECIALIST Lexicon[4]. The Metathesaurus provides information about concepts in the system, including its definition, its semantic types, concepts it is related to, etc. Via the Metathesaurus, a user can also request information about specific attributes themselves (e.g., requesting all concepts assigned to a particular semantic type). The Semantic Network contains information about semantic types and their relationships using the inheritance property of the network type hierarchy. Using the Semantic Network, the user may specify queries about a semantic pair and the relationships between the semantic pairs. Any of the three parameters may be left unknown, allowing the user to specify a wide variety of queries. The SPECIALIST Lexicon contains all lexical records for each word or term as syntactic, morphological (i.e., structure), and orthographic information. Lexical entries may be single or multi-word terms and include information concerning syntactic category, inflectional vocabulary, and allowable complementation patterns.

7.4.1 Clavin User Model Construction. The use of the interface agent in the Clavin system is different here in that the user does not perform explicit actions *per se* in the environment that are observable. To ascribe user intent, interface agent designers must identify the salient characteristics of a domain environment

---

[4]The component names are capitalized in keeping with the UMLS's method of designating the names.

and specifically determine goals a user is trying to achieve, the reason and/or cause for pursuing those goals, and the actions to achieve those goals. For the Virtual SpacePlane and PESKI domains, the user model (i.e., the goals, actions, and pre-conditions and associated probabilities and utility functions) could be elicited *a priori*. However, the robustness of the CIA architecture and knowledge representation allows us to model the spoken inquiries from the user as well as the return answers to the database queries to effectively help the user.

As Figure 13 shows, the user's inquiry, as represented by the wff, has a hierarchical structure representation within the Bayesian user model. The leaves of the graph are actions and the rest of the nodes are sub-goals, except for the query node, which is a goal node.

To construct proactive queries based on relevant concepts within the user model, the interface agent uses the rank ordering feature of the decision-theoretic CIA architecture. Internally, at each time slice (i.e., observation), the CIA architecture ranks every node in the user model. When the user utters a new inquiry, the interface agent and correction adaptation agents receive notification that a new query has been added to the user model by the wrapper agent. For the Clavin system, a domain-dependent correction adaptation agent was designed to proactively generate new queries based on the user's current inquiry and past inquiries and concepts seen. Taking the rank ordering of the nodes for the most recent time slice, this correction adaptation agent performs a traversal over previous queries to determine which might be candidates for a proactive query. The agent only selects queries with an expected utility greater than the user's autonomous threshold, and which guarantees that resulting proactive query will have an expected utility greater than the user's autonomous threshold. This approach insures the interface agent will make this query autonomously. After candidate queries have been selected, the children of these queries are considered for inclusion in the proactive query. The chosen children are then further processed to determine which of their children are processed. There

133

are several considerations when generating the proactive query. If a parent was selected for the proactive query, at least one child will be, since the expected utilities are added and normalized for AND nodes and the maximum expected utility child is selected for OR nodes.

7.4.2 Related Work.   The Microsoft Office '95 Answer Wizard (Heckerman and Horvitz 1998) takes a Bayesian perspective on information retrieval for inferring the goals and needs of users. The approach centers on the resource-intensive, declarative construction of probabilistic knowledge bases (Bayesian networks) for interpreting user queries. The authors state

> people typically are unfamiliar with the terms that expert users or software designers may use to refer to structures displayed in a user interface, states of data structures, and classes of software functionality.

The authors assume that the order of terms in a query is irrelevant and terms not available in their lexicon are ignored. Furthermore, the author's assume *term independence*. Clavin does not make any of these assumptions. As the authors contend, these assumptions may lead to significant information loss, given the importance of structure and dependencies among words in human communication. The authors are extending their research by looking into ways to automatically construct the Bayesian networks used.

7.4.3 Issues.   The following issues arise within the context of the Clavin System.

1. **"On-the-Fly" User Model Construction:** The CIA architecture's decision-theoretic approach offers a framework with which to determine which sub-queries are relevant and can be combined to make proactive queries on behalf of the user. By using the expected utilities of the various sub-query concepts (e.g., isA(x, dog), color(y, red), action(hit(john, ball))), the interface agent can combine those sub-queries with high expected utility. This approach

134

allows the agent to combine concepts in meaningful, non-ad hoc fashion. This procedural-based user model construction method allows the CIA architecture to begin with no domain knowledge and incrementally construct the user model as the user interacts with Clavin. This unanticipated side benefit allows designers to mitigate the classic knowledge acquisition bottleneck problem.

2. **Agent Autonomy:** As discussed in the definitions on agent requirements, autonomy alone perhaps best defines agency. Since Clavin interacts with many heterogenous information sources, some of them dynamic (e.g., news streams), the interface agent component of Clavin should be free to autonomously fetch new information, reason over it, and present updated results to the user. The issue becomes the following: how much autonomy should the agent have?

3. **Dynamic Information Sources** Unlike the Microsoft Office '95 Answer Wizard, Clavin is presented with dynamic, possibly massive information sources. Therefore a declarative approach to user model construction is not feasible. The sub-query relevance approach taken in Clavin offers a useful alternative to the declarative approach as well as data-centric, statistical learning approaches.

7.5 Conclusion

This chapter presented three disparate domains — a virtual space plane, a probabilistic expert system shell, and a natural language query information management system. These domains are most assuredly *not* toy domains. The diversity of the three domains and the integration of the CIA architecture into these domains has shown the robustness of the architecture. These domains have also served to show the usefulness of the decision-theoretic approach to ascribing user intent used by the CIA architecture. Results indicate the CIA architecture is an efficient, effective, and extensible architecture for user intent ascription. The architecture is capable of providing assistance that is tailored to individual users' needs. This assistance can be adapted to the dynamic needs and goals of users over time.

VIII. Future Research

There are a number of existing issues raised by this research and these issues are fruitful areas for future research. The items are presented in order of expected benefit to the CIA architecture.

1. **Computational Cognitive Architecture** — In order to dynamically calculate the various human factors, an accurate computational cognitive model of the various human factors is required. In this dissertation, several human factors were chosen and utility functions were calculated for these. As a result of the lack of a computational cognitive architecture, all but one of the human factors were static and provided *a priori* in the user's profile. The problem of representing the dynamics involved with all the human factors was beyond the scope of this research. However, the CIA architecture supports the dynamic calculation of the human factors. The dynamic calculation of workload is a testament to this support. Future work needs to focus on an improved architecture that can calculate the value of all the human factors.

2. **Learning User Utility Models** — The approach used in this research was to specify the user's utility models was an expert systems approach similar to the one presented by Horvitz and Rutledge (1991). Automated learning of the utility functions, such as the methods discussed by Rust (1996), would ensure more accurate, dynamic utility models.

3. **IaDEA Implementation** — As mentioned in Chapter VI, the Interface agent Development Environment Architecture (IaDEA) is not yet implemented. To ease future integration of the CIA architecture into other domains and to enhance the architecture's use in the three existing domains, IaDEA must be realized. As a first step towards realizing IaDEA, it should be developed to support the existing CIA architecture, requirements, metrics, and correction adaptation agents.

136

4. **Interactive User Models** — In the implementation of the interface agent's user model, a user may view the Bayesian network user model graphically. As a result of viewing the user model, the user develops a better model of the interface agent's model of the user. This action however occurs only at the explicit request of the user and the graphical display is static and non-interactive. Kay (1996) describes the usefulness of providing an interactive user model. In particular, she states the main advantage is allowing the user to collaboratively help the agent elicit information about the user. Kay realizes this advantage by providing users the um toolkit. The um toolkit is usable by a wide diversity of users because of its flexibility in the kinds of models it can represent. These models range from a simple list of user model component names (with descriptors and explanations) to elicitation tools and graphical viewers.

Within the Core Interface Agent architecture, providing users an interactive, visual user model is beneficial for several reasons. First, the interactive user model allows for increased collaboration between the user and agent and, as a result, a more accurate (as measured by the requirements utility function) user model. Secondly, a user can indicate which goal he/she is pursuing explicitly versus the interface agent attempting to recognize the user's goal via keyhole plan recognition. For example, a user could click on the goal node. The interface agent would then perform the actions associated with the goal. Thirdly, and related to the second benefit, a user can view the actions associated with a goal, possibly correcting misconceptions and misunderstandings about which actions must be performed to achieve a goal. Fourth, viewing the agent's user model solidifies the user's model of the interface agent's user model. Finally, the user can correct an incorrect user model.

5. **Detecting User Misconceptions** — The interface agent could offer assistance for misconceptions. The CIA architecture possesses several artifacts that

would allow this enhancement to be implemented. The CIA architecture stores a rank ordering of the goals based on expected utility. If user performs actions outside top-ranked goals, then we have one of the above two situations. Using the *misconception metric* defined in Equation (17) as an indicator that a misconception exists, the interface agent can engage in a correction bidding process with correction adaptation agents capable of improving this metric. Currently these correction adaptation agents do not exist.

6. **Correction Model Improvements** — Another improvement could leverage the work in multi-agent system cooperation work. In particular, the correction model presented in Chapter V along with the associated Σ and Π (see Appendix C) assume simple interaction between correction adaptation agents. Specifically, the agents are what is termed "level-0" agents. That is, they have no knowledge about other agents, only their own internal knowledge about their own actions and "world." There exists numerous researchers investigating various levels of agents and the methods for modeling the other agents (Gmytrasiewicz 1996; Zeng and Sycara 1996; Noh and Gmytrasiewicz 1997; Haynes, Sen, Schoenefeld, and Wainwright 1995; Haynes and Sen 1997; Müller 1996; Vidal and Durfee 1997; Parkes and Ungar 1997). For example, Zeng and Sycara (1996) present a market-based multi-agent system where the agents use a Bayesian belief updating to learn to negotiate based on what an agent "believes" the other agents might bid in the market.

IX. Conclusions

While GESIA was a first step towards realizing the SIRDS goal, the research presented in this dissertation has served to greatly advance this goal. This research has successfully tested the hypothesis of providing an effective, efficient, and extensible uncertainty-based architecture for user intent ascription. While certain aspects of the approach are addressed by other research, no one has taken a synergistic approach combining the three most prominent research fields in interface agent research — artificial intelligence, human-computer interaction, and user modeling. Validation of the larger framework hypothesis within which this work was performed, a holistic, decision-theoretic methodology for the interface agent development life cycle, was beyond the scope of this research due to the immense size of the problem. However, in addition to the CIA architecture, several advances were made in this arena. In particular, the in-depth analysis of human-computer interaction, artificial intelligence, and user modeling interface agent research identified the strengths and weaknesses of these three fields as well as relevant interface agent research in these fields. Furthermore, an investigation of tools to support the development of interface agents uncovered major deficiencies in these environments, including a lack of environment specification, adaptivity mechanisms, and knowledge base and reasoning mechanisms. These deficiencies were addressed in the intelligent agent development environment. ASLAN supports the existing CIA architecture by providing guidance to interface agent developers for what is important to model within the target system environment. The reuse of the correction adaptation agents and reasoning mechanism and the knowledge-based systems approach supports rapid prototyping of interface agents into new application domains.

The CIA architecture addresses the need for effective, efficient, and extensible uncertainty-based architectures for user intent ascription. The use of "probability modules" (Winston 1984) is an *ad hoc* approach to ascribing user intent. Fur-

139

thermore, these modules do not readily support the addition and deletion of rules. The Bayesian network-based user model handles the inherent ambiguity in ascribing user intent. Furthermore, the network's directed, acyclic graph structure is a good choice for capturing the causal relationship between pre-conditions, goals, and actions. Where purely statistical correlation approaches only consider the likelihood of offering assistance to users, the decision-theoretic approach used by the CIA architecture also considers the utility in offering this assistance. The user's utility model explicitly models the affect human factors have on the user's decision making process. This model allows the interface agent to account for the reasons (i.e., why) a user chooses an action in pursuit of a goal.

The keyhole plan recognition approach for user intent ascription is valid and has been used in other related systems (e.g., Albrecht, Zukerman, Nicholson, and Bud (1997) and Waern (1996)). This dissertation improves on their work by using a decision-theoretic approach which allows the interface agent to determine both the likelihood a user is pursuing a goal as well as the user's desire for assistance. Additionally, since the interface agent can offer collaborative assistance, the user is able to explicitly confirm or deny the actual benefit of the assistance offered. This user interaction with his/her user model, to include the ability to set assistance thresholds and an agent's proactiveness, ensures the user remains in control of the assistance offered. With regards to research using a decision-theoretic approach to offering assistance to users (e.g., Breese and Heckerman (1996), Horvitz and Barry (1995), Horvitz and Rutledge (1991) and Karagiannidis, Koumpis, and Stephanidis (1996)), the research presented in this dissertation offers the benefit of user model correction, to include dynamic, procedural-based construction of the user model. The approach used by all the cited references is to construct the user model *a priori* with the use of expert knowledge. The research presented in this dissertation has shown that the user models may be constructed "on the fly," greatly reducing the

knowledge acquisition bottleneck. This construction method is absolutely necessary when dealing with environments with voluminous, dynamic information sources.

The correction model along with the set of requirements and associated metrics for measuring the interface agent's capability in meeting the requirements prevents the "mindless" corrections to the user model that would have little or no impact, or possibly adversely affect the interface agent's performance. Existing approaches to modifying user models over time suffer from two problems. The first is an inability to determine when and how to correct the user model. The CIA architecture's correction model focuses on corrections that have the greatest impact on improving the user model. Second, existing systems have limited methods (typically only one) of adapting the user model. The CIA architecture has no theoretical limitations on the number of correction adaptation agents that may be included in the bidding process. One of the most fruitful areas for future research following this dissertation is an in-depth investigation into correction adaptation agent evolution. Additionally, as mentioned previously, the correction model can function not only to maintain the user model but to acquire new knowledge as well.

The experiments performed and results obtained served two main purposes. First, the experiments and results showed the CIA architecture is an effective, efficient, and extensible architecture for user intent ascription. The CIA architecture can offer assistance that is beneficial to a wide diversity of users based on the users preferences, biases, application usage, etc. The architecture is robust in that it can be used for widely disparate domains. The side benefit of the decision-theoretic approach, a procedural-based knowledge acquisition approach, is one of the most rewarding results of the research performed. Second, these experiments and results have shown several key areas for future research. The ability to provide a computational cognitive architecture as well as the ability to learn the user's utility models should greatly improve the CIA architecture's utility for providing assistance.

Appendix A. Publications

In addition to the motivating publications listed in Section 1.1, a number of publications are a direct product of my research. The conference topics are diverse, ranging from artificial intelligence to autonomous agents to training systems. This diversity is a testament to the broad contribution the research presented in this dissertation represents. The publications are as follows:

- The principle of symbiotic information retrieval and decision support as an approach for collaborative, decision-theoretic assistance was presented at the 19th Interservice/Industry Training Systems and Education Conference (Banks, Stytz, Santos Jr., and Brown 1997). The paper was nominated for best conference paper.

- The use of utility functions for intelligent decision making in user models for interface agents was presented at the Twelfth Canadian Conference on Artificial Intelligence (Brown, Santos Jr., and Banks 1998b).

- The correction model and the concept of user model adaptation via a multi-agent system was presented at the Second International Conference on Autonomous Agents (Brown, Santos Jr., Banks, and Oxley 1998).

- The proposal for an integrated development environment to support the developmental life cycle of interface agents was presented at the AAAI-98 workshop on software tools for agent development (Brown, Santos Jr., Banks, and Stytz 1998a).

- The integration of the CIA architecture into PESKI was presented at the AAAI-98 Spring Symposium Workshop on Interactive and Mixed-Initiative Decision-Theoretic Systems (Brown, Santos Jr., and Banks 1998a).

- The integration of the CIA architecture into the virtual space plane was presented at the AAAI-98 Spring Symposium Workshop on Intelligent Environments (Brown, Santos Jr., Banks, and Stytz 1998b).

Appendix B. Human Factor Attribute Assessment

A user's profile captures the static human factor, such as skill. These factors are determined off-line and are assumed static throughout the interaction. However, the CIA architecture supports update of the static human factors, if necessary. The difference between a static and dynamic human factor is that static human factors are not re-assessed at each time increment, whereas dynamic human factors are.

Each human factor is scaled with a real number between 0 to 1, 1 indicating the best (e.g., *expertise* = 1 indicates an extremely skilled user, *temporal memory* = 0 means the user forgets everything). This number is, in a sense, a measure of the human's ability with respect to the human factor.

The following static human factors were used by the CIA architecture:

- **Expertise**

- **Operative Memory**

- **Temporal Memory**

- **Spatial Memory**

- **Response Time**

For the experiments in this dissertation, workload was the only dynamic human factor. Numerous researchers have attempted to define workload within the confines of a specific domain. One of the goals of the interface agent is to reduce the workload on the user. Therefore, the more actions the interface agent performs for the user, the more user's workload is reduced.

A user's workload is assessed as follows:

$$user\ work = \frac{|user\ actions|}{|user\ actions + agent\ actions|}$$

$$time = \frac{user\ time}{user\ time + agent\ time}$$

144

$$user\ workload \ = \ \frac{user\ work}{time}.$$

Appendix C. Correction Model Protocol and Strategy

Chapter V stated the bidding process correction model is adopted from Müller (1996). The negotiation protocol function Π and negotiation strategy function Σ introduced in Definition 5 were implemented to model a sealed-bid, single award strategy. The two functions (adapted from Müller (1996, pg. 253)) are given as follows:

Definition 7 *Let the negotiation protocol function be $\Pi = (K, \pi : A \times K \mapsto 2^K)$. Let $A = \{IA, CA_1, \ldots, CA_n\}$ be a set of agents where IA represents the interface agent and CA_i represents a correction adaptation agent. Let t be the bid deadline time in seconds and b be the bid. Let π be defined as:*

$$\pi(a, k) = \begin{cases} \{ANNOUNCE(t)\} & \text{if } a = IA, \ k = start, \\ \{BID(b_i)\} & \text{if } a = CA_i, \ k = ANNOUNCE(t), \\ \{REJECT(b_i), ACCEPT(b_i)\} & \text{if } a = IA, \ k = BID(b_i), \\ \{done\} & \text{if } a = IA, \ k = REPORT(b_i), \\ \{done\} & \text{if } a = CA_i, \ k = REJECT(b_i) \end{cases}$$

Definition 8 *Let the negotiation strategy function be $\Sigma = \{\sigma_i : \Pi \times A \times K \times 2^D \times U \mapsto K \times N | 1 \leq i \leq k\}$. Let $\sigma_i(\Pi, a_i, k, N, u_i) = (k', N')$ with $k' \in \pi(a_i, k)$, $N' \subseteq N$. Σ is defined as follows[1]:*

$$\sigma_{IA}(start, N, u_{IA}) = (ANNOUNCE(t), [u_{IA}, 1])$$

$$\sigma_{CA_i}(ANNOUNCE(t), N, u_{CA_i}) = \{BID(u_{CA_i}), [u_{CA_i}, 1]\}$$

$$\sigma_{IA}(BID(u_{CA_i}), N, u_{IA}) = \text{if bids\_received} = n \text{ or elapsed\_time} > t \text{ then}$$
$$ACCEPT(\lfloor N \rfloor) \text{ to } CA_i \text{ with bid}(\lfloor N \rfloor) \text{ and}$$
$$REJECT(\lfloor N \rfloor) \text{ to } CA_j, j \neq i$$

---

[1] *When the protocol Π and agent $a \in A$ are understood, they are omitted.*

$$if\ u_{CA_i} > \lfloor N \rfloor\ then\ N = [u_{CA_i}, 1];$$

$$\sigma_{CA_i}(REJECT(u_{CA_j}), N, u_{CA_j}) = (done, N), i \neq j$$

$$\sigma_{CA_i}(ACCEPT(b_i), N, u_{CA_j}) = (REPORT(u_{CA_i}), N)$$

$$\sigma_{CA_i}(REPORT(b_i), N, u_{CA_i}) = (done, N).$$

Semantically, π and σ_i can be summarized as follows: at the start of the bid process, the interface agent announces the bid deadline, t, to all the correction adaptation agents. Each correction adaptation agent makes a bid, b_i. A bid is rejected outright if it will not improve the interface agent's user model. The best bid is accepted and all others are rejected. When the bid has been accomplished, the winning correction adaptation agents reports back to the interface agent.

Appendix D. Correction Adaptation Agent Implementation

The CIA architecture allows for any number of correction adaptation agents. For the research performed in this dissertation, a total of thirteen correction adaptation agents were implemented. These agents were chosen for both their ease of implementation (several of the agents differ from one another by only a few lines of C++ code) and their usefulness to alter the user model parameters and structure. The correction adaptation agents implemented are given in Table 4.

Table 4: Correction Adaptation Agents Implemented for PESKI and Clavin

Name	Description
BKD Probability Update	Updates the probabilities in the conditional probability tables by using the Bayesian Knowledge Discoverer (BKD) data mining tool (Ramoni and Sebastiani 1997).
Query Update Agent	This agent was used by Clavin only. It adds new queries from the user to the interface agent user model. It also adds proactive queries as described in Chapter VII.

Table 4: Correction Adaptation Agents Implemented for PESKI and Clavin

Name	Description
History Fading	Three separate agents adapt the history fading. One increases or decreases the number of observations stored in the evaluator history stack based on the simple reinforcement learning. Two other agents perform history fading: one for increasing and one for decreasing the number of observations. These are simpler agents (as compared with the agent using reinforcement learning). History fading is similar to the way other researcher fade observations (Foner 1994; Waern 1996); however, they do *not* adapt the length of the fading. These fading functions have precedence over the observation fading function (detailed next). That is, if the interface agent allows, for example, five observations via the observation fading function, but the history fading function allows only one observation, the agent "forgets" observations older than the last one.
Periodic Update Fading	Three separate agents adapt the update period. Recall the interface agent performs periodic belief updating if no new evidence arrives from the target application. These three faders alter the periodicity of the update. As with the history fading agents, one agent uses simple reinforcement learning and two other agents are used to perform an increase and decrease, respectively, of the periodicity.

Table 4: Correction Adaptation Agents Implemented for PESKI and Clavin

Name	Description
Cut-off Time Fading	Three agents update the cut-off time, akin to the history fading and period update agents. That is, the agents increase or decrease the amount of time the interface agent has to offer assistance per Equation (10).
Bid Deadline Cut-off Time Fading	Three agents alter the bid deadline. These agents either increase or decrease the amount of time the correction adaptation agents have to respond to a bid per Equation (15). The shorter the deadline the quicker the interface agent can reward the best bidder. However, some correction adaptation agents may not be able to bid in time, affecting the correction adaptation quantity metric (see Equation (14)).
Metric History Fading	Increases or decreases the number of time slices included in the calculation of the metrics, and therefore requirements utility function. If the metrics are calculated over only the most recent time slice, the metrics and requirements utility function capture how well the interface agent met its requirements for the last suggestion. If the metrics are calculated over the entire history, this measures how well the interface agent has met its requirements over this history.

Fading functions are used to "forget" past evidence. Two different fading functions were used — one for wall clock time, one for length (no more than N observations). For each fading function, we have one of two possible actions: increase

or decrease the fading function. We divide our state space differently for each fading function. For wall clock time, the state space is divided into seconds; for the length, the state space is divided by integer lengths.

Appendix E. Bayesian Network Experiment Definition

This appendix contains the definition of the two Bayesian networks described in Chapter VII and shown in Figure 7. The upper Bayesian network in Figure 7 is defined by *variables* g1, g2, a1, a2, a3, p1,and p2. The lower Bayesian network is defined by *variables* g3, g4, a4, a5, a6, p3, and p4.

```
// Bayesian Network in the Interchange Format
// Produced by BayesianNetworks package in JavaBayes
// Output created Fri Aug 28 08:58:06 EDT 1998
// Bayesian network
network Internal-Network{
//14 variables and 14 probability distributions
}
variable g1{//2 values
type discrete[2] { False True };
property position = (201, 253) ;
}
variable g2{//2 values
type discrete[2] { False True };
property position = (117, 267) ;
}
variable a2{//2 values
type discrete[2] { False True };
property position = (136, 143) ;
}
variable a1{//2 values
type discrete[2] { False True };
property position = (227, 147) ;
}
variable a3{//2 values
type discrete[2] { False True };
property position = (45, 139) ;
}
variable p1{//2 values
type discrete[2] { False True };
property position = (192, 119) ;
}
variable p2{//2 values
type discrete[2] { False True };
```

```
property position = (69, 103) ;
}
variable g3{//2 values
type discrete[2] { False True };
property position = (407, 379) ;
}
variable a4{//2 values
type discrete[2] { False True };
property position = (444, 455) ;
}
variable a5{//2 values
type discrete[2] { False True };
property position = (348, 467) ;
}
variable a6{//2 values
type discrete[2] { False True };
property position = (271, 472) ;
}
variable g4{//2 values
type discrete[2] { False True };
property position = (314, 365) ;
}
variable p3{//2 values
type discrete[2] { False True };
property position = (428, 287) ;
}
variable p4{//2 values
type discrete[2] { False True };
property position = (311, 263) ;
}
probability ( g1 a1 a2 p1 ) { //4 variable(s) and 16 values
table 0.99 0.5 0.83 0.33 0.67 0.83 0.5 0.01
0.01 0.5 0.17 0.67 0.33 0.17 0.5 0.99 ;
}
probability ( g2 a2 a3 p2 ) { //4 variable(s) and 16 values
table 0.99 0.83 0.5 0.33 0.5 0.33 0.83 0.01
0.01 0.17 0.5 0.67 0.5 0.67 0.17 0.99 ;
}
probability ( a2 ) { //1 variable(s) and 2 values
table 0.65 0.35 ;
}
probability ( a1 ) { //1 variable(s) and 2 values
```

```
table 0.3 0.7 ;
}
probability ( a3 ) { //1 variable(s) and 2 values
table 0.25 0.75 ;
}
probability ( p1 ) { //1 variable(s) and 2 values
table 0.45 0.55 ;
}
probability ( p2 ) { //1 variable(s) and 2 values
table 0.55 0.45 ;
}
probability ( g3 p3 ) { //2 variable(s) and 4 values
table 0.95 0.05 0.05 0.95 ;
}
probability ( a4 g3 ) { //2 variable(s) and 4 values
table 0.95 0.05 0.05 0.95 ;
}
probability ( a5 g3 g4 ) { //3 variable(s) and 8 values
table 0.99 0.67 0.33 0.01 0.01 0.33 0.67 0.99 ;
}
probability ( a6 g4 ) { //2 variable(s) and 4 values
table 0.9 0.01 0.1 0.99 ;
}
probability ( g4 p4 ) { //2 variable(s) and 4 values
table 0.99 0.01 0.01 0.99 ;
}
probability ( p3 ) { //1 variable(s) and 2 values
table 0.45 0.55 ;
}
probability ( p4 ) { //1 variable(s) and 2 values
table 0.55 0.45 ;
}
```

Bibliography

Akoulchina, I. and J.-G. Ganascia (1997). SATELIT-Agent: An adaptive interface based on learning interface agents technology. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 21–32. Vienna, New York: Springer Wien New York. Available from http://um.org.

Albrecht, D. W., I. Zukerman, A. E. Nicholson, and A. Bud (1997). Towards a Bayesian model for keyhole plan recognition in large domains. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 365–376. Vienna, New York: Springer Wien New York. Available from http://um.org.

Ambrosini, L., V. Cirillo, and A. Micarelli (1997). A hybrid architecture for user-adapted information filtering on the World Wide Web. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 59–61. Vienna, New York: Springer Wien New York. Available from http://um.org.

Bacchus, F. (1993, July). Using first-order probability logic for the construction of Bayesian networks. In D. Heckerman and A. Mamdani (Eds.), *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pp. 219–226.

Baecker, R. M., J. Grudin, W. A. S. Buxton, and S. Greenberg (1995). From customizable systems to intelligent agents. In *Readings in Human-Computer Interaction: Toward the Year 2000* (Second ed.)., Chapter 12, pp. 783–792. Morgan Kaufmann.

Balch, T. (1997a, July). Learning roles: Behavioral diversity in robot teams. In *Collected Papers from the 1997 AAAI Workshop on Multiagent Learning*, pp. 7–12. AAAI Press. AAAI Technical Report WS-97-03.

Balch, T. (1997b). Social entropy: a new metric for learning multi-robot teams. In *Proceedings of the 10th FLAIRS Conference (FLAIRS-97)*.

Banks, S. B., R. A. Harrington, E. Santos Jr., and S. M. Brown (1997, May). Usability testing of an intelligent interface agent. In *Proceedings of the Sixth International Interfaces Conference (Interfaces 97)*, pp. 121–123.

Banks, S. B. and C. S. Lizza (1991, June). Pilot's Associate: A cooperative, knowledge-based system application. *IEEE Expert*, 18–29.

Banks, S. B., M. R. Stytz, E. Santos Jr., and S. M. Brown (1997, December). User modeling for military training: Intelligent interface agents. In *Proceedings of the 19th Interservice/Industry Training Systems and Education Conference*, pp. 645–653.

Barbuceanu, M. and M. S. Fox (1996a). The architecture of an agent building shell. In M. J. Woolridge, J. P. Müller, and M. Tambe (Eds.), *Intelligent Agents II: Agent Theories, Architectures, and Languages*, pp. 235–250. Berlin: Springer.

Barbuceanu, M. and M. S. Fox (1996b). The design of a coordination language for multi-agent systems. In *Proceedings of the Agent Theories, Architectures, and Languages Conference*, pp. 341–355.

Barker, D. (1998, March/April). Secret agent man: IBM turns over a new leaf with Ginkgo. *PC AI 12*(2), 21–22.

Bauer, M. (1996). Acquisition of user preferences for plan recognition. In A. Jameson, C. Paris, and C. Tasso (Eds.), *Proceedings of the Fifth Internataional Conference on User Modeling (UM '96)*, pp. 105–112. SpringerWien, New York.

Bawcom, D. (1997). An incompleteness handling methodology for validation of bayesian knowledge bases. Master's thesis, Air Force Institute of Technology.

Bellman, R. (1957). *Dynamic Programming*. Princeton, New Jersey: Princeton University Press.

Benyon, D. and D. Murray (1993, December). Adaptive systems: from intelligent tutoring to autonomous agents. *Knowledge-Based Systems 6*(4), 197–219.

Billinghurst, M. and J. Savage (1996). Adding intelligence to the interface. In *Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium*, pp. 168–176. Piscataway, NJ: IEEE Press.

Blahut, R. E. (1987). *Principles and Practice of Information Theory*. Addison-Wesley.

Bollacker, K. D., S. Lawrence, and C. L. Giles (1998, May 9–13). CiteSeer: An autonomous web agent for autonomous retrieval and indentification of interesting publication. In K. P. Sycara and M. Woolridge (Eds.), *Proceedings of the Second International Conference on Autonomous Agents (Agents '98)*, pp. 116–123. ACM Press.

Boone, G. (1998, May 9–13). Concept features in Re:Agent, an intelligent email agent. In K. P. Sycara and M. Woolridge (Eds.), *Proceedings of the Second International Conference on Autonomous Agents (Agents '98)*, pp. 141–148. ACM Press.

Breese, J. S. and D. Heckerman (1996). Decision-theoretic troubleshooting: A framework for repair and experiment. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI–96)*, Portland, Oregon, pp. 124–132.

Brown, S. M., R. A. Harrington, E. Santos Jr., and S. B. Banks (1997, June). User models, interface agents and expert systems. In *Proceedings of the Embedding User Models in Intelligent Applications Workshop*, pp. 12–17. held in conjunction with the Sixth International Conference on User Modeling (UM '97).

Brown, S. M., E. Santos Jr., and S. B. Banks (1997, May). A dynamic Bayesian intelligent interface agent. In *Proceedings of the Sixth International Interfaces Conference (Interfaces 97)*, pp. 118–120.

Brown, S. M., E. Santos Jr., and S. B. Banks (1998a, March). Supporting incremental knowledge elicitation in decision-theoretic systems. In *Proceed-*

ings of the 1998 AAAI Spring Symposium on Interactive and Mixed-Initiative Decision-Theoretic Systems, pp. 14–15.

Brown, S. M., E. Santos Jr., and S. B. Banks (1998b, June). Utility theory-based user models for intelligent interface agents. In *Proceedings of the Twelfth Canadian Conference on Artificial Intelligence (AI '98)*, pp. 378–392.

Brown, S. M., E. Santos Jr., and S. B. Banks (1999, January). Intelligent interfaces for decision-theoretic systems. In *Proceedings of the Intelligent User Interfaces Conference (IUI '99)*. (submitted to).

Brown, S. M., E. Santos Jr., S. B. Banks, and M. E. Oxley (1998, May). Using explicit requirements and metrics for interface agent user model correction. In *Proceedings of the Second International Conference on Autonomous Agents (Agents '98)*, pp. 1–7.

Brown, S. M., E. Santos Jr., S. B. Banks, and M. R. Stytz (1998a, July). IaDEA: A development environment architecture for building generic intelligent user interface agents. In *Proceedings of the AAAI-98 Workshop on Software Tools for Agent Development*.

Brown, S. M., E. Santos Jr., S. B. Banks, and M. R. Stytz (1998b, March). Intelligent interface agents for intelligent environments. In *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*. AAAI Press, AAAI Technical Report SS-98-02.

Bull, S. (1997). See Yourself Write: A simple student model to make students think. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 315–326. Vienna, New York: Springer Wien New York. Available from http://um.org.

Chauhan, D. and A. D. Baker (1998, May 9–13). JAFMAS: A multiagent application development system. In K. P. Sycara and M. Woolridge (Eds.), *Proceedings of the Second International Conference on Autonomous Agents (Agents '98)*, pp. 100–107. ACM Press.

Chen, L. and K. Sycara (1998, May 9–13). WebMate: A personal agent for browsing and searching. In K. P. Sycara and M. Woolridge (Eds.), *Proceedings of the Second International Conference on Autonomous Agents (Agents '98)*, pp. 132–140. ACM Press.

Chickering, D. M., D. Heckerman, and C. Meek (1997). A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, San Francisco, CA, pp. 80–89. Morgan Kaufmann Publishers.

Chin, D. N. (1991). Intelligent interfaces as agents. In J. W. Sullivan and S. W. Tyler (Eds.), *Intelligent User Interfaces*. ACM, New York.

Chiu, B. C., G. I. Webb, and M. Kuzmycz (1997). A comparison of first-order and zeroth-order induction for Input-Output Agent Modelling. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 347–358. Vienna, New York: Springer Wien New York. Available from http://um.org.

Chu, T. and Y. Xiang (1997). Exploring parallelism in learning belief networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, San Francisco, CA, pp. 90–98. Morgan Kaufmann Publishers.

Coen, M. (1994, June). SodaBot: A software agent environment and construction system. Technical Report 1493, Massachusetts Institute of Technology Artificial Intelligence Laboratory.

Cohen, P. R., A. J. Cheyer, M. Wang, and S. C. Baeg (1994, March). An open agent architecture. In *AAAI Spring Symposium*, pp. 1–8.

Conati, C., A. S. Gertner, K. VanLehn, and M. J. Druzdzel (1997). On-line student modeling for coached problem solving using Bayesian networks. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth*

International Conference, UM97, pp. 231–242. Vienna, New York: Springer Wien New York. Available from http://um.org.

Concordia Home Page (1998). Concordia documentation. World Wide Web Page http://www.meitca.com/HSL/Projects/Concordia/.

Cooke, N. J. (1994). Varieties of knowledge elicitation techniques. *International Journal of Human-Computer Studies 41*(6), 801–849.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence 42*, 393–405.

Corbett, A. T. and A. Bhatnagar (1997). Student modeling in the ACT programming tutor: Adjusting a procedural learning model with declarative knowledge. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 243–254. Vienna, New York: Springer Wien New York. Available from http://um.org.

Cormen, T. H., C. E. Leiserson, and R. L. Rivest (1990). *Introduction to Algorithms*. McGraw-Hill.

Cousins, Steve B., e. a. (1991). CABeN: A collection of algorithms for belief networks. Technical Report WUCS-91-25, Department of Computer Science, Washington University.

Csinger, A., K. S. Booth, and D. Poole (1994). AI meets authoring: User models for intelligent multimedia. *Artificial Intelligence Review 8*, 447–468.

Csinger, A. and D. Poole (1996). User models and perceptual salience: Formal abduction for model recognition and presentation design. In *Proceedings of the Fifth International Conference on User Modeling (UM '96)*, pp. 51–58.

Cypher, A. (1991). Eager: Programming repetitive tasks by example. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, pp. 33–39.

Davis, A. M. (1993). *Software Requirements: Objects, Functions & States*. P T R Prentice Hall.

Decker, K. S., A. Pannu, K. Sycara, and M. Williamson (1997, February). Designing behaviors for information agents. In W. L. Johnson (Ed.), *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, pp. 404–413.

DeWitt, R. (1995). Vagueness, semantics, and the language of thought.

Doux, A.-C., J.-P. Laurent, and J.-P. Nadal (1997). Symbolic data analysis with the K-means algorithm for user profiling. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 359–361. Vienna, New York: Springer Wien New York. Available from http://um.org.

Draper, D. L. and S. Hanks (1994). Localized partial evaluation of belief networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pp. 170–177.

Druzdzel, M. J. (1997). Five useful properties of probabilistic knowledge representations from the point of view of intelligent systems. *Fundamenta Informaticae 30*(3/4), 241–254. Special issue on Knowledge Representation and Machine Learning.

Druzdzel, M. J. and L. C. van der Gaag (1995). Elicitation of probabilities for belief networks: Combining qualitative and quantitative information. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, Montreal, Quebec, Canada, pp. 141–148.

Etzioni, O. and D. Weld (1994). A softbot-based interface to the internet. *Communications of the ACM 37*(7), 72–76.

Fano, A. E. (1998, May). SHOPPER'S EYE: Using location-based filtering for a shopping agent in the physical world. In K. P. Sycara and M. Woolridge (Eds.), *Proceedings of the Second International Conference on Autonomous Agents (Agents '98)*, pp. 416–421. ACM Press.

161

Fink, J., A. Kobsa, and A. Nill (1997). Adaptable and adaptive information access for all users, including the disabled and the elderly. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 171–173. Vienna, New York: Springer Wien New York. Available from http://um.org.

Foner, L. N. (1994, June). Paying attention to what's important: Using focus of attention to imporve unsupervised learning. Master's thesis, Massachusetts Institute of Technology.

Franklin, S. and A. Graesser (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag.

Friedman, N. and M. Goldszmidt (1996). Learning Bayesian networks with local structure. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, Portland, Oregon, pp. 252–262.

Gavrilova, T. and A. Voinov (1997, June). An approach to mapping of user model to corresponding interface parameters. In *Proceedings of the Embedding User Models in Intelligent Applications Workshop*, pp. 24–29. held in conjunction with the Sixth International Conference on User Modeling (UM '97).

Geiger, D. and D. Heckerman (1995). A characterization of the Dirichlet distribution with application to learning Bayesian networks. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, Montreal, Quebec, Canada, pp. 196–207.

General Magic (1998). Odyssey home page. World Wide Web Page http://www.genmagic.com/technology/odyssey.html.

Genesereth, M. R. and R. E. Fikes (1997). *Knowledge Interchange Format Version 3.0 Reference Manual.* Interlingua Working Group of the DARPA Knowledge Sharing Effort.

Gertner, A. S., C. Conati, and K. VanLehn (1998). Procedural help in Andes: Generating hints using a Bayesian network student model. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, San Francisco, CA. Morgan Kaufmann Publishers.

Giangrandi, P. and C. Tasso (1997). Managing temporal knowledge in student modeling. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 415–426. Vienna, New York: Springer Wien New York. Available from http://um.org.

Gmytrasiewicz, P. (1996). On reasoning about other agents. In M. J. Woolridge, J. P. Müller, and M. Tambe (Eds.), *Intelligent Agents II: Agent Theories, Architectures, and Languages*, pp. 143–155. Berlin: Springer.

Goldberg, D. and M. J. Matarić (1997, July). Interference as a tool for designing and evaluating multi-robot controllers. In *Proceedings of the Fourteenth National Conference on Artifical Intelligence (AAAI 97)*, pp. 637–642.

Goldman, R. P. and E. Charniak (1993). A language for construction of belief networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*(3), 196–208.

Gonzalez, A. J. and D. D. Dankel (1993). *The Engineering of Knowledge-Based Systems: Theory and Practice*. Prentice-Hall, Inc.

Goodwin, R. (1993). Formalizing properties of agents. Technical Report CMU-CS-93-159, Carnegie Mellon Institute.

Gori, M., M. Maggini, and E. Martinelli (1997). Web-browser access through voice input and page interest prediction. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 17–19. Vienna, New York: Springer Wien New York. Available from http://um.org.

163

Gray, R., D. Kotz, G. Cybenko, and D. Rus (1997). Agent Tcl. In W. Cockayne and M. Zyda (Eds.), *Itinerant Agents: Explanations and Examples with CD-ROM*. Manning Publishing. Imprints by Manning Publishing and Prentice Hall.

Gray, R. S. (1995a, December). Agent Tcl: A transportable agent system. In *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95)*.

Gray, R. S. (1995b, December). *Agent Tcl: Alpha Release 1.1*. Available by WWW at http://www.cs.dartmouth.edu/~rgray/documentation/doc.1.1.ps.gz.

Greiner, R., A. J. Grove, and D. Schuurmans (1997). Learning Bayesian nets that perform well. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, San Francisco, CA, pp. 198–207. Morgan Kaufmann Publishers.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition 5*(2), 199–220.

Gutkauf, B., S. Thies, and G. Domik (1997). A user-adaptive chart editing system based on user modeling and critiquing. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 159–170. Vienna, New York: Springer Wien New York. Available from http://um.org.

Ha, V. and P. Haddawy (1997). Problem-focused incremental elicitation of multi-attribute utility models. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, Providence, Rhode Island, pp. 215–222.

Haddawy, P. (1994). Generating bayesian networks from probability logic knowledge bases. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pp. 262–269.

Han, E.-H. S., D. Boley, M. Gini, R. Gross, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore (1998, May). WebACE: A web agent for document categorization and exploration. In K. P. Sycara and M. Woolridge (Eds.), *Proceedings of the Second International Conference on Autonomous Agents (Agents '98)*, pp. 408–415. ACM Press.

Harrington, R. A., S. Banks, and E. Santos Jr. (1996a). Development of an intelligent user interface for a generic expert system. In M. Gasser (Ed.), *Online Proceedings of the Seventh Midwest Artificial Intelligence and Cognitive Science Conference.*

Harrington, R. A., S. Banks, and E. Santos Jr. (1996b). GESIA: Uncertainty-based reasoning for a generic expert system intelligent user interface. In *Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence*, pp. 52–55.

Harrington, R. A. and S. M. Brown (1997). Intelligent interface learning with uncertainty. In E. Santos Jr. (Ed.), *Proceedings of the Eighth Midwest Artificial Intelligence and Cognitive Science Conference*, pp. 27–34. AAAI Press.

Haynes, T. and S. Sen (1997). Crossover operators for evolving a team. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo (Eds.), *Genetic Programming 1997: Proceedings of the Second Annual Conference.* MIT Press.

Haynes, T., S. Sen, D. Schoenefeld, and R. Wainwright (1995, November). Evolving a team. In E. V. Siegel and J. R. Koza (Eds.), *Working Notes for the AAAI Symposium on Genetic Programming*, Cambridge, MA. AAAI.

Heckerman, D. (1995). A Bayesian approach to learning causal networks. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, Montreal, Quebec, Canada, pp. 274–284.

Heckerman, D. and D. Geiger (1995). Learning Bayesian networks: A unification for discrete and Gaussian domains. In *Proceedings of the Eleventh Annual Con-*

ference on Uncertainty in Artificial Intelligence (UAI–95), Montreal, Quebec, Canada, pp. 285–295.

Heckerman, D. and E. Horvitz (1998). Inferring informational goals from free-text queries: A Bayesian approach. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, San Francisco, CA. Morgan Kaufmann Publishers.

Henrion, M. (1989). Some practical issues in constructing belief networks. In L. Kanal, T. Levitt, and J. Lemmer (Eds.), *Uncertainty in Artificial Intelligence 3*, pp. 161–173. North Holland: Elsevier Science Publishers B.V.

Hirst, G., C. DiMarco, E. Hovy, and K. Parsons (1997). Authoring and generating health-education documents that are tailored to the needs of the individual patient. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 107–118. Vienna, New York: Springer Wien New York. Available from http://um.org.

Höök, K. (1997, March). Steps to take before IUI becomes real. Presented at the Workshop "The Reality Of Intelligent Interface Technology".

Horvitz, E. (1997). Agents with beliefs: Reflections on Bayesian methods for user modeling. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 441–442. Vienna, New York: Springer Wien New York. Available from http://um.org.

Horvitz, E. and M. Barry (1995). Display of information for time-critical decision making. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, Montreal, Quebec, Canada, pp. 296–305.

Horvitz, E., J. Breese, D. Heckerman, D. Hovel, and K. Rommelse (1998). The Lumiére project: Bayesian user modeling for inferring goals and needs of software users. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, San Francisco, CA. Morgan Kaufmann Publishers.

Horvitz, E. and G. Rutledge (1991, July). Time-dependent utility and action under uncertainty. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 151–158. Morgan Kaufmann.

IBM (1997). IBM agent building environment (ABE): A toolkit for building intelligent agent applications. World Wide Web Page http://www.networking.ibm.com/iag/iagsoft.htm.

IBM Corp. (1998a). IBM aglets software development kit. World Wide Web Page http://www.trl.ibm.co.jp/aglets/.

IBM Corp. (1998b). IBM intelligent agents Ginkgo white paper. World Wide Web Page http://www.networking.ibm.com/iag/iaginkgo.htm.

Jameson, A. (1996). Numeric uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interactions 5*, 193–251.

Jameson, A., C. Paris, and C. Tasso (Eds.) (1997). *Preface of User Modeling: Proceedings of the Sixth International Conference, UM97*. Springer Wien New York.

Järvnin, P. (1993, March). Notes on assumptions of user modelling. Technical Report A-1993-2, University of Tampere, Department of Computer Science, P.O. Box 607, SF-33101 Tampere, Finland.

Kalyuga, S., P. Chandler, and J. Sweller (1997). Levels of expertise and user-adapted formats of instructional presentations: A cognitive load approach. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 261–272. Vienna, New York: Springer Wien New York. Available from http://um.org.

Karagiannidis, C., A. Koumpis, and C. Stephanidis (1996, August). Deciding 'what', 'when', 'why', and 'how' to adapt in intelligent multimedia presentation systems. In G. Faconti and T. Rist (Eds.), *Proceedings of the Twelfth European Conference on Artificial Intelligence Workshop "Towards a Standard Reference*

Model for Intelligent Multimedia Presentation Systems". John Wiley & Sons, Ltd.

Kay, J. (1994). Lies, damn lies, and stereotypes: pragmatic approximations of users. In *User Modeling: Proceedings of the Fourth International Conference, UM94*, pp. 175–184. Early draft of invited keynote presented at UM'94 available at http://www.cs.su.oz.au/ judy/Research/index.html.

Kay, J. (1996). The um toolkit for reusable, long term user models. *User Modeling and User-Adapted Interaction 4*(3), 149–196.

Keates, S. and P. Robinson (1997, June). User performance modeling and cognitive load. In *Proceedings of the RESNA Annual Conference*, pp. 342–344. RESNA Press.

Keeney, R. L. and H. Raiffa (1993). *Decisions with Multiple Objects: Preferences and Value Tradeoffs*. Cambridge University Press.

King, W. J. and J. Ohya (1996). The representation of agents: Anthropomorphism, agency, and intelligence. In *Electronic Proceedings of CHI96*.

Kirman, J., A. Nicholson, M. Lejter, T. Dean, and E. Santos Jr. (1993). Using goals to find plans with high expected utility. In *Proceedings of the Second European Workshop on Planning*, Linkoping, Sweden, pp. 158–170.

Kjaerulff, U. (1994). Reduction of computational complexity in Bayesian networks through removal of weak dependencies. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pp. 374–382.

Kjaerulff, U. (1995). HUGS: Combining exact inference and Gibbs sampling in junction trees. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 368–375.

Koda, T. and P. Maes (1996). Agents with faces: The effects of personification of agents. In *Proceedings of HCI'96*.

Kozierok, R. and P. Maes (1993). A learning interface agent for scheduling meetings. In *Proceedings of the ACM SIGCHI International Workshop on Intelligent User Interfaces*, pp. 81–88.

Lashkari, Y., M. Metral, and P. Maes (1994). Collaborative interface agents. In *Proceedings of the National Conference on Artificial Intelligence*, Cambride, MA. MIT Press.

Lauritzen, S. L. and D. J. Spiegelhalter (1988). Local computations with probabilities on graphical structures and their applications to expert systems. *J. Royal Statistical Society 50*(2), 157–224.

Lewis, M. (1998). Designing for human-agent interaction. *AI Magazine 19*(2), 67–78.

Lieberman, H. (1994). A user interface for knowledge acquisition from video. In A. Cypher (Ed.), *Conference of the American Association for Artificial Intelligence*. MIT Press.

Lieberman, H. (1995). Letizia: An agent that assists web browsing. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Lieberman, H. (1997). Autonomous interface agents. In *Proceedings of the Computer-Human Interaction (CHI) Conference*, pp. 67–74. ACM Press.

Linden, G., S. Hanks, and N. Lesh (1997). Interactive assessment of user preference models: The Automated Travel Assistant. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 67–78. Vienna, New York: Springer Wien New York. Available from http://um.org.

Lowry, M. and R. Duran. *The Handbook of Artificial Intelligence*, Volume IV, Chapter XX: Knowledge-based Software Engineering, pp. 243–322. Addison-Wesley Publishing Company, Inc.

Maes, P. (1994a, July). Agents that reduce work and information overload. *Communications of the ACM 37*(7), 811–821.

Maes, P. (1994b). Modeling adaptive autonomous agents. *Artificial Life Journal 1*(1 & 2). MIT Press (C. Langton, Ed.).

Maes, P. and R. Kozierok (1993). Learning interface agents. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), Washington, DC.*

Maglio, P. P. and R. Barrett (1997). How to build modeling agents to support web searchers. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 5–16. Vienna, New York: Springer Wien New York. Available from http://um.org.

Malone, T., K. Lai, R. Rao, and D. Rosenblitt (1989). The Information Lens: An intelligent system for information sharing and coordination. In M. Olson (Ed.), *Tecnological Support for Working Group Collaboration.* Lawrence Erlbaum Associates.

Martin, D. L., A. Cheyer, and G. L. Lee (1996, April). Agent development tools for the open agent architecture. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, pp. 387–404. The Practical Application Company Ltd.

Mayfield, J., Y. Labrou, and T. Finin (1996). Evaluation of KQML as an agent communication language. In M. J. Woolridge, J. P. Müller, and M. Tambe (Eds.), *Intelligent Agents II: Agent Theories, Architectures, and Languages*, pp. 347–360. Berlin: Springer.

Meek, C. and D. Heckerman (1997). Structure and parameter learning for causal independence and causal interaction models. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, San Francisco, CA, pp. 366–375. Morgan Kaufmann Publishers.

Metral, M. E. (1993, May). Design of a generic learning interface agent. Bachelor of Science, Massachussets Institute of Technology.

Michael H. Coen (mhcoen@ai.mit.edu) (1998, May). Re: Sodabot. E-mail to Scott Brown (sbrown777@acm.org).

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag.

Microsoft (1997). Microsoft agent. Available at http://www.microsoft.com/intdev/agent/.

Mitchell, T., R. Caruana, D. Freitag, J. McDermott, and D. Zabowski (1994, July). Experience with a learning personal assistant. *Communications of the ACM 37*(7), 81–91.

Mitsubishi Electric Information Technology Center America (1997, April). Concordia: An infrastructure for collaborating mobile agents. In *Proceedings of the First International Workshop on Mobile Agents 97 (MA '97), Berlin, Germany.*

Mulgund, S. S. and G. L. Zacharias (1996, August). A situation-driven adaptive pilot/vehicle interface. In *Proceedings of the Third Annual Symposium on Human Intercation with Complex Systems*, pp. 193–198.

Müller, J. P. (1996, August). A cooperation model for autonomous agents. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings (Eds.), *Intelligent Agents III: Agent Theories, Architectures, and Languages*, ECAI'96 Workshop (ATAL), pp. 245–260. Springer.

Murphy, M. and M. McTear (1997). Learner modelling for intelligent CALL. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 301–312. Vienna, New York: Springer Wien New York. Available from http://um.org.

Myers, B. A. (1996, December). A brief history of human computer interaction technology. Technical Report CMU-CS-96-163, Carnegi Mellon University.

Nicholson, A. E. and J. M. Brady (1994). Dynamic belief networks for discrete monitoring. *IEEE Transactions on Systems, Man, and Cybernetics 24*(11), 1593–1610.

Noh, S. and P. J. Gmytrasiewicz (1997). Agent modeling in antiair defense: A case study. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 389–400. Vienna, New York: Springer Wien New York. Available from http://um.org.

Ntuen, C. A. (1997, August). A formal method for deriving command production language from human intents. In *Proceedings of the Seventh International Conference on Human-Computer Interaction (HCI International '97)*.

Oard, D. W. and G. Marchionini (1996, May). A conceptual framework for text filtering. Technical Report CS-TR-3643, University of Maryland, College Park, MD.

ObjectSpace, Inc. (1998). Voyager technical white papers. World Wide Web Page $http://www.objectspace.com/voyager/technical_white_papers.html$.

Paranagama, P., F. Burstein, and D. Arnott (1997). Modelling the personality of decision makers for active decision support. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 79–81. Vienna, New York: Springer Wien New York. Available from http://um.org.

Parkes, D. C. and L. H. Ungar (1997, July). Learning and adaptation in multiagent system. In *Collected Papers from the 1997 AAAI Workshop on Multiagent Learning*, pp. 47–52. AAAI Press. AAAI Technical Report WS-97-03.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

Pestello, H. F. G. and F. P. Pestello (1991). Ignored, neglected, and abused: The behavior variable in attitude-behavior research. *Symbolic Interaction 14*(3), 341–351.

Petrie, C. (1996, December). Agent-based engineering, the web, and intelligence. *IEEE Expert 11*(6), 24–29.

Pohl, W. (1997, August). LaboUr – machine learning for user modeling. In *Proceedings of the Seventh International Conference on Human-Computer Interaction (HCI International '97)*, pp. 27–30.

Pohl, W. and J. Höhle (1997). Mechanisms for flexible representation and use of knowledge in user modeling shell systems. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 403–414. Vienna, New York: Springer Wien New York. Available from http://um.org.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence 64*(1), 81–129.

Pradhan, M., M. Henrion, G. Provan, B. D. Favero, and K. Huang (1995). The sensitivity of belief networks to imprecise probabilities: An experimental investigation. Technical Report KSL–95–66, Knowledge Systems Laboratory.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning 1*, 81–106.

Ramoni, M. and P. Sebastiani (1997). Learning Bayesian networks from incomplete databases. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, San Francisco, CA, pp. 401–408. Morgan Kaufmann Publishers.

Reticular Systems, I. (1998). AgentBuilder white paper. World Wide Web Page http://www.agentbuilder.com/Documentation/WhitePaper/index.html.

Rhodes, B. J. (1996, April). Remembrance agent: A continuously running automated information retrieval system. In *Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology, London, UK*, pp. 487–495.

Rich, E. (1983). Users are individuals: Individualizing user models. *International Journal of Man-Machine Studies 18*, 199–214.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorenson (1991). *Object-Orient Modeling and Design.* Prentice Hall.

Rust, J. (1996). Do people really behave Bellman's principle of optimality? Available via anonymous FTP $ftp : //gemini.econ.yale.edu/pub/John\_Rust/Bellman/bell.ps$. Yale University, (unpublished).

Sánchez, J. A., F. S. Azevedo, and J. J. Leggett (1995, June). PARAgente: Exploring the issues in agent-based user interfaces. In *Proceedings of the First International Conference on Multiagent Systems – ICMAS '95*, pp. 320–327.

Santos Jr., E., D. O. Banks, and S. B. Banks (1997). MACK: A tool for acquiring consistent knowledge under uncertainty. In *Proceedings of the AAAI Workshop on Verification and Validation of Knowledge-Based Systems*, pp. 23–32.

Santos Jr., E., H. T. Gleason, and S. B. Banks (1997). BVAL: Probabilistic knowledge-base validation. In *Proceedings of the AAAI Workshop on Verification and Validation of Knowledge-Based Systems*, pp. 13–22.

Sarkar, S. and I. Murthy (1996). Constructing efficient belief network structures with expert provided information. *IEEE Transactions on Knowledge and Data Engineering 8*(1), 134–143.

Schäfer, R. and T. Weyrath (1997). Assessing temporally variable user properties with dynamic Bayesian networks. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 377–388. Vienna, New York: Springer Wien New York. Available from http://um.org.

Selker, T. (1994, July). Coach: A teaching agent that learns. *Communications of the ACM 37*(7), 92–99.

Shimony, S. E., C. Domshlak, and J. Santos, Eugene (1997). Cost-sharing in Bayesian knowledge bases. In *Proceedings of the Thirteenth Annual Conference*

on Uncertainty in Artificial Intelligence (UAI-97), San Francisco, CA, pp. 421–428. Morgan Kaufmann Publishers.

Shoham, Y. (1993, March). Agent-oriented programming. *Artificial Intelligence 60*(1), 51–92.

Shoham, Y. (1997). Conditional utility, utility independence, and utility networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, San Francisco, CA, pp. 429–436. Morgan Kaufmann Publishers.

Small, J. M. H. R. L. (1995). An intelligent interface in an associate system. *Human/Technology Interaction in Complex Systems 7*, 1–44.

Spragins, J. D., J. L. Hammond, and K. Pawlikowski (1991). *Telecommunications: Protocols and Design.* Addison Wesley.

Stallman, R. (1997, July). *GNU Emacs Manual, Thirteenth Edition, Updated for Emacs Version 20.1.* Free Software Foundation. ISBN 1-882114-06-X.

Stanfill, C. and D. Waltz (1986, December). Toward memory-based reasoning. *Communications of the ACM 29*, 1213–1228.

Stanford University Center for Design Research (1998). JATLite home page. World Wide Web Page http://java.stanford.edu/.

Stary, C. (1997, August). The role of interaction modeling in future cognitive ergonomics: Do interaction models lead to formal specifications of involved machine intelligence? In *Proceedings of the Seventh International Conference on Human-Computer Interaction (HCI International '97).*

Stefanuk, V. L. (1997, June). Embedding user models in intelligent interfaces. In *Proceedings of the Embedding User Models in Intelligent Applications Workshop*, pp. 18–23. held in conjunction with the Sixth International Conference on User Modeling (UM '97).

Stein III, D. J., S. B. Banks, E. Santos Jr., and M. L. Talbert (1997). Utilizing goal-directed data mining for incompleteness repair in knowledge bases. In

E. Santos Jr. (Ed.), *Proceedings of the Eighth Midwest Artificial Intelligence and Cognitive Science Conference*, pp. 82–85. AAAI Press.

Stytz, M. R. and S. B. Banks (1997, December). The virtual spaceplane: A modeling and simulation tool for advanced prototyping, requirements development, and training for the manned spaceplane project. In *Proceedings of the 19th Interservice/Industry Training Systems and Education Conference*.

Sun Microsystems, I. (1998). Java technology home page. World Wide Web Page http://www.javasoft.com/.

Sycara, K., K. Decker, A. Pannu, M. Williamson, and D. Zeng (1996, December). Distributed intelligent agents. *IEEE Expert 11*(6), 36–46.

Terveen, L. G. (1995, April–June). Overview of human-computer collaboration. *Knowledge-Based Systems 8*(2–3), 67–81.

Thomas, C. G. (1993, December). Design, implementation and evaluation of an adaptive user interface. *Knowledge-Based Systems 6*(4), 230–238.

Van Veldhuizen, D. A., G. B. Lamont, and E. Santos Jr. (1998, April). Comparing computer generated military actors with specific skills. In *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE) Aerospace/Defense Sensing and Controls Workshop on Modeling and Simulating Sensory Response for Real and Virtual Environments*.

Vassileva, J. (1997). A new view of interactive human-computer environments. In A. Jameson, C. Paris, and C. Tasso (Eds.), *User Modeling: Proceedings of the Sixth International Conference, UM97*, pp. 433–435. Vienna, New York: Springer Wien New York. Available from http://um.org.

Vidal, J. M. and E. H. Durfee (1997, July). Agents learning about agents: A framework and analysis. In *Collected Papers from the 1997 AAAI Workshop on Multiagent Learning*, pp. 71–76. AAAI Press. AAAI Technical Report WS-97-03.

Waern, A. (1996). *Recognising Human Plans: Issues for Plan Recognition in Human-Computer Interaction.* Ph. D. thesis, Royal Institute of Technology.

Waern, A. (1997, March). What is an intelligent interface? notes from an introduction seminar.

Williams, E. M. (1997). *Modeling Intelligent Control of Distributed Cooperative Inferencing.* Ph. D. thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology.

Winston, P. H. (1984). *Artificial Intelligence, 2d Ed.* Addison-Wesley.

Wooldridge, M. and N. R. Jennings (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review 10*(2), 115–152. Available at ftp.elec.qmw.ac.uk/pub/isag/distributed-ai/publications/KE-REVIEW-95.ps.Z.

Yoshida, K. (1997, August). User modeling by graph-based induction. In *Proceedings of the Seventh International Conference on Human-Computer Interaction (HCI International '97)*, pp. 23–26.

Yoshida, K. and H. Motoda (1997). Automated user modeling for intelligent interface. *International Journal of Human-Computer Interaction 8*(3), 237–258.

Zeng, D. and K. Sycara (1996, August). How can an agent learn to negotiate? In J. P. Müller, M. J. Wooldridge, and N. R. Jennings (Eds.), *Intelligent Agents III: Agent Theories, Architectures, and Languages*, ECAI'96 Workshop (ATAL), pp. 233–244. Springer.

Vita

Scott Brown was born in Chambersburg, Pennsylvania, in May, 1969. After moving to Duluth, Minnesota during high school, he graduated from the University of Minnesota-Duluth with a Bachelor of Computer Engineering in 1992. He was commissioned into the United States Air Force in March 1992; his first tour was at Eglin AFB, Fort Walton Beach, Florida. While stationed at Eglin AFB, he earned his Master of Science in Computer Science (Software Engineering option). Immediately following award of his Master degree in 1995, he was selected to obtain his doctorate at the Air Force Institute of Technology. His next assignment is to the Human Effectiveness Directorate of the Air Force Research Laboratory on Wright-Patterson AFB, OH.

Permanent address: 5901 Coralberry Drive
Clayton, OH 45315